

GIT TANPA DRAMA

Belajar Praktis Version Control Untuk Pemula



Dian Hanifudin Subhi
Usman Nurhasan
Putra Prima Arhandi
Sofyan Noor Arief

GIT

TANPA DRAMA

**Belajar Praktis Version Control
untuk Pemula**

**Dian Hanifudin Subhi
Usman Nurhasan
Putra Prima Arhandi
Sofyan Noor Arief**



PENERBIT KBM INDONESIA

adalah penerbit dengan misi memudahkan proses penerbitan buku-buku penulis di tanah air Indonesia, serta menjadi media *sharing* proses penerbitan buku.

GIT TANPA DRAMA

Belajar Praktis Version Control untuk Pemula

Copyright @ 2026 By Dian Hanifudin Subhi, dkk

All right reserved

Penulis

Dian Hanifudin Subhi

Usman Nurhasan

Putra Prima Arhandi

Sofyan Noor Arief

Desain Sampul

Eca Art

Tata Letak

Husnud Diniyah

Editor

Dr. Muhamad Husein Maruapey, Drs., MSc.

Background isi buku di ambil dari [https://www. freepik. com/](https://www.freepik.com/)

Official

Depok, Sleman-Jogjakarta (Kantor)

Penerbit KBM Indonesia

Anggota IKAPI/No. IKAPI 279/JTI/2021

081357517526 (Tlpn/WA)

Website

[https://penerbitkbm. com](https://penerbitkbm.com)

[www. penerbitbukumurah. com](http://www.penerbitbukumurah.com)

Email

[naskah@penerbitkbm. com](mailto:naskah@penerbitkbm.com)

Distributor

[https://penerbitkbm. com/toko-buku/](https://penerbitkbm.com/toko-buku/)

Youtube

Penerbit KBM Sastrabook

Instagram

@penerbit. kbmindonesia

@penerbitbukujogja

ISBN: 978-634-278-173-9

Cetakan ke-1, Februari 2026

14,8 x 21 cm, x + 168 halaman

Isi buku diluar tanggungjawab penerbit
Hak cipta merek KBM Indonesia sudah terdaftar di DJKI-
Kemenkumham dan isi buku dilindungi undang-undang.

Dilarang keras menerjemahkan, memfotokopi, atau
memperbanyak sebagian atau seluruh isi buku ini
tanpa seizin penerbit karena beresiko sengketa hukum

Sanksi Pelanggaran Pasal 113

Undang-Undang No. 28 Tahun 2014 Tentang Hak Cipta

- i. 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 Rp 100. 000. 000 (seratus juta rupiah).
- ii. 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 Rp 500. 000. 000,00 (lima ratus juta rupiah).
- iii. 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 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 Rp 1. 000. 000. 000,00 (satu miliar rupiah).
- iv. 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 Rp 4. 000. 000. 000,00 (empat miliar rupiah).

Kata Pengantar

Git tidak harus sulit, tidak harus penuh drama, tidak harus ketakutan mengetik `git push`. Buku ini lahir dari pengalaman mengajar pemula yang frustrasi dengan Git. Mereka sering bertanya: "Kenapa Git begitu membingungkan?" Jawabannya: bukan Git yang rumit, tapi cara kita mengajarkannya yang sering terlalu teknis, terlalu cepat, dan terlalu banyak jargon.

Git Tanpa Drama berbeda. Buku ini ditulis dengan prinsip:

- **Praktik dulu, teori kemudian**
- **Bahasa sederhana, straightforward**
- **Windows-friendly:** Semua instruksi untuk Windows dengan Git Bash
- **Error itu normal:** Setiap bab punya "Kesalahan Umum", belajar dari kesalahan

Git adalah skill yang akan terus Anda pakai sepanjang karier sebagai developer. Investasi waktu Anda hari ini akan terbayar ribuan kali lipat. Mari mulai perjalanan tanpa drama ini.

Malang, 30 Januari 2026

Tim Penulis

Git Tanpa Drama

Pengantar

Pernahkah Anda memiliki folder bernama `tugas-akhir`, `tugas-akhir-final`, `tugas-akhir-final-eneran`, dan `tugas-akhir-final-beneran-v2`? Atau mungkin Anda pernah menghapus kode yang sebenarnya masih dibutuhkan, tapi sudah terlambat karena tidak ada backup? Kalau iya, berarti Anda sangat membutuhkan Git.

Git adalah sistem version control yang membantu Anda melacak setiap perubahan pada kode atau dokumen. Bayangkan Git seperti mesin waktu untuk proyek Anda, setiap perubahan tersimpan rapi, bisa dilihat kapan saja, bahkan bisa dikembalikan kalau ada kesalahan.

Untuk Siapa Buku Ini?

Buku ini ditulis khusus untuk Anda yang:

- **Pemula** dalam menggunakan Git
- Bekerja dengan **Windows** sebagai sistem operasi utama
- Ingin belajar **praktis** tanpa tenggelam dalam teori yang rumit
- Sering merasa bingung atau takut ketika menggunakan Git
- Ingin berkolaborasi dengan tim menggunakan **GitHub**

Isi Buku

Dalam buku ini, Anda akan mempelajari Git secara bertahap, dari nol hingga mahir berkolaborasi:

1. **Fondasi Git**: Setup, commit pertama, melihat riwayat perubahan

2. **Branching & Merging:** Bekerja di cabang terpisah dan menggabungkannya dengan aman
3. **Kolaborasi GitHub:** Upload proyek, sinkronisasi dengan tim, dan workflow Pull Request
4. **Troubleshooting:** Cara membatalkan kesalahan dan menyelesaikan masalah umum

Setiap bab: konsep ringkas + praktik CLI + VS Code + kesalahan umum + latihan.

Yang Membuat Buku Ini Berbeda

Buku ini tidak hanya mengajarkan command Git, tapi juga:

- **Menjelaskan KENAPA**, bukan hanya CARA
- **Fokus Windows:** Tutorial disesuaikan untuk pengguna Windows
- **Git Bash + VS Code:** Kombinasi CLI dan GUI untuk fleksibilitas maksimal
- **GitHub CLI:** Kenalkan tools modern seperti `gh` untuk produktivitas
- **Proyek praktis:** Semua latihan menggunakan satu proyek berkelanjutan (`web-tanpa-drama`)
- **Anti drama:** Strategi menghindari konflik dan kesalahan umum pemula

Cara Menggunakan Buku Ini

1. **Ikuti urutan bab.** Setiap bab membangun pemahaman dari bab sebelumnya.
2. **Praktikkan langsung.** Jangan hanya membaca, ketik setiap command dan amati hasilnya.

3. **Jangan takut salah.** Git dirancang untuk membantu Anda recover dari kesalahan.
4. **Bookmark Lampiran A & B.** Cheat sheet dan glosarium akan sering Anda butuhkan.

Konvensi dalam Buku

- **Code block dengan \$:** menandakan command yang dijalankan di Git Bash

```
$ git status
```

- **Code block tanpa \$:** menandakan output dari command

```
On branch main  
nothing to commit, working tree clean
```

- **[Gambar x.x]:** placeholder untuk diagram atau screenshot
- **Catatan penting:** ditandai dengan blok khusus

Catatan

Ini adalah contoh catatan penting yang perlu Anda perhatikan.

Repo Latihan

Sepanjang buku ini, kita akan membangun proyek sederhana bernama **web-tanpa-drama**, sebuah website HTML sederhana. Proyek ini akan terus berkembang dari bab ke bab, memberi Anda pengalaman praktis yang koheren.

Referensi dan Sumber

Buku ini disusun berdasarkan dokumentasi resmi Git, GitHub, dan best practices dari komunitas developer global. Semua materi

telah diverifikasi dengan sumber-sumber terpercaya termasuk:

- Git Official Documentation (git-scm.com)
- GitHub Docs
- Atlassian Git Tutorials
- Conventional Commits specification

Daftar lengkap referensi tersedia di bagian **Daftar Pustaka** di akhir buku.

Siap Memulai?

Mari kita mulai perjalanan belajar Git tanpa drama. Setelah selesai membaca buku ini, Anda akan percaya diri menggunakan Git untuk proyek pribadi maupun berkolaborasi dengan tim. Selamat belajar!

Kata Pengantar	iii
----------------	-----

Pengantar	v
-----------	---

I FONDASI GIT	1
----------------------	----------

1 Peta Besar Git Tanpa Drama	3
2 Setup Windows: Git Bash, VS Code, GitHub CLI	15
3 Repo Pertama: init & status	25
4 Commit yang Rapi: add & commit	35
5 Membaca Perubahan: log & diff	49
6 Repo Bersih: .gitignore	57

II BRANCHING & MERGING	69
-----------------------------------	-----------

7 Branch untuk Pemula	71
8 Merge Tanpa Drama	83
9 Konflik Merge: resolve & verifikasi	93

III KOLABORASI DENGAN GITHUB	107
-------------------------------------	------------

10 Dari Lokal ke GitHub: remote & push	109
11 Kolaborasi: fetch/pull & push ditolak	115
12 Pull Request Workflow	123

IV ADVANCED & TROUBLESHOOTING	131
--	------------

13 Stash + Pengetahuan Rebase vs Merge	133
14 Undo & Troubleshooting	141

V LAMPIRAN	151
-------------------	------------

A Cheat Sheet	153
B Glosarium	159

Daftar Pustaka	163
Tentang Penulis	166

FONDASI GIT



1. Peta Besar Git Tanpa Drama

1.1 Kenapa Penting

Sebelum terjun ke command-command Git, kita perlu memahami peta besarnya dulu. Bayangkan Anda belajar mengemudi, sebelum gas dan rem, Anda perlu tahu kemana tujuannya, apa fungsi setir, dan mengapa perlu pakai sabuk pengaman. Bab ini adalah “sesi pengenalan mobil” sebelum Anda benar-benar menyetir. Dengan memahami gambaran besar, Anda tidak akan tersesat saat mempraktikkan Git nanti.

1.2 Tujuan Belajar

- Jelaskan masalah yang diselesaikan Git
- Pahami: working directory, staging, repository, remote
- Mengerti alur: edit → stage → commit → push
- Cek instalasi Git

1.3 Konsep Kunci

Drama Git yang Sering Terjadi

Coba ingat-ingat, apakah Anda pernah mengalami hal-hal ini?

Folder Final Syndrome: Anda punya folder skripsi-final, skripsi-final-rev, skripsi-final-BENERAN. Bingung mana yang versi terbaru? Lupa perubahan apa yang ada di setiap versi?

Takut Menghapus: Ada kode yang Anda rasa tidak perlu, tapi takut dihapus karena “siapa tahu nanti butuh lagi.” Akhirnya kode di-comment dan dibiarkan mengotori file.

Kolaborasi Berantakan: Bekerja dengan teman lewat flash disk atau Google Drive. File bentrok, perubahan teman tertimpa perubahan Anda, atau malah ada dua versi berbeda yang bingung cara gabungnya.

Backup Manual: Setiap kali mau coba sesuatu yang berisiko, Anda copy-paste seluruh folder sebagai backup. Folder proyek jadi besar dan berantakan.

Semua masalah di atas adalah "drama" yang bisa dihindari dengan Git.

Git: Mesin Waktu untuk Code

Git adalah sistem **version control** yang diciptakan oleh Linus Torvalds pada tahun 2005. Awalnya dibuat untuk mengelola pengembangan kernel Linux yang melibatkan ribuan kontributor dari seluruh dunia, Git kini menjadi standar industri untuk version control. Popularitasnya didukung oleh platform seperti GitHub, GitLab, dan Bitbucket yang menjadikan kolaborasi software development lebih mudah dari sebelumnya.



Gambar 1.1: *Logo Git - sistem version control yang digunakan oleh jutaan developer di seluruh dunia*

Fungsi Git seperti mesin waktu yang merekam setiap perubahan pada file proyek. Setiap kali Anda menyimpan snapshot (disebut **commit**), Git merekam:

- File apa yang berubah

- Siapa yang mengubah
- Kapan diubah
- Kenapa diubah (melalui pesan commit)

Git bersifat **distributed**, artinya setiap developer memiliki salinan lengkap repository di komputer mereka. Ini berbeda dengan sistem version control terpusat (seperti SVN) di mana semua orang bergantung pada satu server pusat. Dengan Git, Anda tetap bisa bekerja dan commit meski offline, lalu sinkronisasi saat online kembali.

Keuntungan Git dalam Software Development

Selain menyelesaikan drama-drama version control, Git memberikan keuntungan besar dalam pengembangan software profesional:

1. Deploy Versi Berbeda ke Server

Dengan Git, Anda bisa dengan mudah deploy versi aplikasi yang berbeda ke server yang berbeda. Misalnya, versi 1.2.0 di server production, versi 1.3.0-beta di server staging, dan versi terbaru di server development. Cukup `git checkout` ke versi yang diinginkan dan deploy. Tidak perlu menyimpan banyak folder berbeda di server.

2. Deployment yang Lebih Mudah dan Aman

Proses deployment menjadi sesederhana `git pull` di server. Anda tahu persis perubahan apa yang akan diterapkan (via `git diff`), bisa rollback instan kalau ada masalah (via `git checkout` atau `git revert`), dan semua riwayat deployment tercatat otomatis.

3. Continuous Integration/Deployment (CI/CD)

Git adalah fondasi dari pipeline CI/CD modern. Setiap kali Anda push commit, bisa otomatis trigger proses testing, build, dan deployment. Tools seperti GitHub Actions, GitLab CI, atau Jenkins semua bekerja berdasarkan Git.

4. **Code Review yang Terstruktur**

Dengan Pull Request, setiap perubahan kode bisa direview oleh tim sebelum masuk ke production. Ini mengurangi bug, meningkatkan kualitas kode, dan membantu transfer knowledge antar developer.

5. **Eksperimen Tanpa Risiko**

Ingin coba refactoring besar? Atau test library baru? Buat branch baru, coba sesuka hati. Kalau gagal, tinggal hapus branch. Kalau sukses, merge ke branch utama. Kode production tetap aman.

6. **Audit Trail Lengkap**

Setiap perubahan tercatat dengan detail: siapa, kapan, dan kenapa. Ini penting untuk compliance, debugging, dan memahami evolusi codebase. Kalau ada bug di production, Anda bisa trace kembali commit mana yang memperkenalkan bug tersebut.

7. **Hotfix Cepat**

Ada bug kritis di production? Buat branch hotfix dari versi production, fix bug, deploy, lalu merge kembali. Semua bisa dilakukan tanpa mengganggu development fitur baru yang sedang berjalan di branch lain.

Komponen Utama

1. **Working Directory** (Meja Kerja Anda)

Tempat Anda bekerja sehari-hari. Di sini Anda edit file, tambah file baru, atau hapus file. Perubahan yang Anda buat di sini belum tercatat dalam Git. Analogi: Kertas-kertas di meja kerja Anda. Masih berantakan, masih bisa diubah-ubah.

2. **Staging Area** (Keranjang Siap Simpan)

Tempat Anda menyiapkan perubahan yang akan disimpan. Tidak semua file yang berubah harus masuk staging, Anda bisa pilih mana yang siap. Analogi: Keranjang belanja. Anda pilih barang mana yang mau dibeli (di-commit), sisanya dikembalikan ke rak.

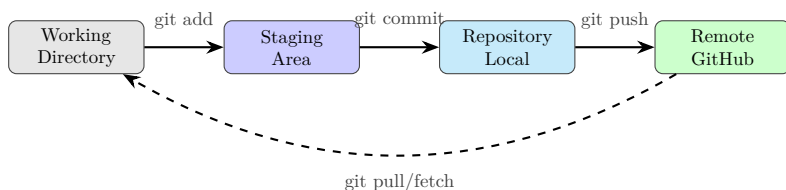
3. **Repository** (Brankas/Arsip Permanen)

Tempat Git menyimpan semua riwayat perubahan. Setiap commit tersimpan di sini. Repository lokal ada di komputer Anda. Analogi: Lemari arsip yang rapi. Setiap dokumen punya label, tanggal, dan catatan.

4. **Remote Repository** (Brankas di Cloud)

Repository yang ada di server online seperti GitHub. Fungsinya untuk backup dan kolaborasi dengan orang lain. Analogi: Safety deposit box di bank. Anda dan tim bisa akses dari mana saja.

Workflow git seperti yang terlihat pada Gambar1.2 diawali dengan Working Directory (edit) → Staging Area (git add) → Repository Local (git commit) → Remote GitHub (git push). Garis putus-putus menunjukkan sinkronisasi kembali dengan git pull/fetch.



Gambar 1.2: *Workflow Git lengkap*

Branch & Merge

Branch adalah jalur kerja terpisah. Bayangkan Anda menulis novel, tapi mau coba ending alternatif tanpa merusak naskah utama. Anda bikin salinan, coba-coba di sana. Kalau bagus, gabungkan ke naskah utama. Itulah branch.

Merge adalah proses menggabungkan perubahan dari satu branch ke branch lain. Git sangat pintar dalam menggabungkan perubahan, bahkan kalau dua orang edit file yang sama (selama tidak edit baris yang sama persis).

Pull Request (PR)

Pull Request adalah cara formal untuk mengusulkan perubahan di GitHub. Anda bilang ke tim, "Hei, saya sudah selesai fitur X di branch ini, tolong review dan gabungkan ke branch utama". Tim bisa beri komentar, minta perubahan, atau langsung setuju. Ini mekanisme kerja profesional yang membuat kolaborasi tetap rapi dan terkendali.

1.4 Praktik: Cek Instalasi

1. Buka Git Bash pada start menu Windows. Maka anda akan melihat terminal command line.

```
dian@LAPTOP MINGW64 ~  
$
```

Jangan panik melihat layar hitam. Ini adalah jendela "command line" tempat kita berbicara langsung ke Git.

2. Cek Versi Git dengan mengetik command `git --version`

```
$ git --version
```

```
git version 2.42.0.windows.1
```

Kalau Git sudah terinstall, Anda akan melihat output `git version` diikuti oleh angka. Angka versinya mungkin berbeda, yang terpenting menampilkan info versi. Jika muncul error seperti `command not found`, berarti Git belum terinstall. Silahkan simak pembahasan instalasi di Bab 2.

3. Buat Folder Proyek

```
$ cd ~  
$ mkdir web-tanpa-drama  
$ cd web-tanpa-drama  
$ echo "# Web Tanpa Drama" > README.md  
$ ls
```

Folder sudah siap tetapi belum menjadi repository Git (dibahas di Bab 3).

1.5 Praktik VS Code: Orientasi Awal

Sekarang mari kita lihat proyek yang sama dari VS Code, editor yang akan sering kita pakai.

1. Buka VS Code

Buka aplikasi Visual Studio Code. Kalau belum punya, kita akan install lengkap di Bab 2.

2. Buka Folder Proyek

- (a) Pilih **File** → **Open Folder** dari menu
- (b) Cari folder **web-tanpa-drama** yang baru Anda buat
- (c) Klik "Select Folder"

Anda akan melihat file **README.md** di sidebar kiri.

3. Lihat Panel Source Control

Di sidebar kiri, ada ikon percabangan (ikon Git) biasanya posisi ketiga dari atas. Klik ikon itu dan akan muncul pesan: "No source control providers registered". Hal itu disebabkan karena folder kita belum diinisialisasi sebagai repo Git. Kita akan pakai panel ini mulai Bab 3. Anda dapat menggunakan perintah Git atau menggunakan panel Source Control di VS Code (direkomendasikan).

4. Buka Terminal Git Bash di VS Code

VS Code punya terminal built-in yang bisa Anda gunakan tanpa keluar dari editor.

- (a) Pilih **Terminal** → **New Terminal** dari menu atau tekan **Ctrl+`** (backtick)
- (b) Di dropdown terminal (pojok kanan atas panel terminal), pilih "Git Bash"

Sekarang Anda punya Git Bash yang terintegrasi dengan VS Code. Ini sangat memudahkan workflow nanti.

1.6 Kesalahan Umum

Berikut beberapa kesalahan yang sering dilakukan pemula saat pertama kali kenal Git:

1. Langsung belajar command tanpa paham konsep

Akibat: Bingung kenapa harus add dulu baru commit. Kenapa ada push dan pull.

Solusi: Pahami konsep staging, repository, dan remote dulu (seperti yang kita bahas di bab ini).

2. Takut mencoba karena takut “merusak” sesuatu

Akibat: Tidak pernah praktik, hanya baca teori. Git tetap terasa asing.

Solusi: Git dirancang untuk aman. Di lokal, hampir semua kesalahan bisa di-undo. Jadi berani eksperimen.

3. Mengira Git sama dengan GitHub

Akibat: Bingung bedanya git command dengan fitur GitHub.

Solusi: Git = sistem version control yang jalan di komputer Anda. GitHub = platform online untuk hosting repository Git.

4. Berpikir Git hanya untuk programmer

Akibat: Desainer, penulis, atau data scientist merasa Git bukan untuk mereka.

Solusi: Git bisa lacak perubahan file apapun. Banyak penulis pakai Git untuk manuscript, desainer untuk file SVG, dll.

5. Tidak pernah commit karena “belum sempurna”

Akibat: Kerja seharian tapi tidak ada checkpoint. Kalau ada masalah, tidak bisa kembali.

Solusi: Commit itu boleh kecil dan sering. Tidak harus fitur lengkap. Yang penting progres tercatat.

6. Abaikan pesan commit

Akibat: Commit message: “update”, “fix”, “changes”. Tiga bulan kemudian lupa apa yang diubah.

Solusi: Tulis pesan yang menjelaskan **apa** dan **kenapa**.

1.7 Latihan

Sekarang giliran Anda merenungkan drama-drama yang ingin dihindari dan bagaimana Git membantu. Ambil kertas atau buka notepad. Tuliskan 3 "drama" terkait version control yang pernah atau mungkin akan Anda alami. Lalu, tulis cara Git membantu menghindari drama tersebut berdasarkan apa yang sudah Anda pelajari di bab ini. Beberapa contoh drama dapat dilihat pada tabel 1.1.

Drama	Solusi Git
Saya punya 5 folder dengan nama "final" beda-beda, bingung mana yang terbaru	Git menyimpan semua versi dalam satu folder. Tinggal lihat riwayat commit untuk tahu versi terbaru atau kembali ke versi lama.
Takut hapus kode lama karena mungkin butuh lagi nanti	Dengan Git, Anda bisa hapus dengan percaya diri. Kalau perlu lagi, tinggal lihat commit sebelumnya dan ambil kembali.
Bekerja dengan teman lewat email atau flash disk, sering file bentrok atau perubahan hilang	Git + GitHub memungkinkan sinkronisasi otomatis. Perubahan teman dan Anda bisa digabung secara pintar (merge).

Tabel 1.1: *Tabel Contoh Drama dan Solusi Git*

1.8 Ringkasan

Git adalah sistem version control yang membantu Anda:

- Melacak setiap perubahan file
- Kembali ke versi lama kapan saja
- Bekerja sama tanpa chaos
- Bereksperimen dengan aman via branch

Alur kerja Git: **Working Directory** → **Staging Area** →

Repository → Remote Repository

Komponen utama:

- **Working Directory:** Tempat Anda edit file
- **Staging Area:** Tempat file siap di-commit
- **Repository:** Tempat riwayat commit tersimpan
- **Remote:** Repository di server (GitHub)

Di bab berikutnya, kita akan setup Git, VS Code, dan GitHub CLI secara lengkap di Windows.

1.9 Cheat Commands

Untuk bab ini, command yang kita pakai sangat dasar:

```
$ git --version          # Cek versi Git yang terinstall
$ cd folder_name        # Pindah ke folder tertentu
$ mkdir folder_name     # Buat folder baru
$ ls                    # Lihat isi folder
$ echo "text" > file.txt # Buat file dengan isi tertentu
```

Ini hanya pemanasan. Command Git yang sebenarnya mulai di Bab 3!

2. Setup Windows: Git Bash, VS Code, GitHub CLI

2.1 Kenapa Bab Ini Penting

Sebelum mulai praktik Git yang sesungguhnya, kita perlu memastikan semua tools sudah terpasang dengan benar di komputer Anda. Setup yang benar di awal akan menghindarkan Anda dari masalah teknis yang membingungkan nanti. Seperti memasak, sebelum mulai, pastikan kompor, panci, dan bumbu sudah siap. Bab ini adalah "persiapan dapur" Anda.

2.2 Tujuan Belajar

- Menginstall Git for Windows dengan konfigurasi yang tepat
- Menginstall dan setup VS Code sebagai editor
- Menginstall GitHub CLI (gh) dan memahami perbedaannya dengan git
- Mengkonfigurasi identitas Git (nama dan email)
- Melakukan autentikasi dengan GitHub
- Memverifikasi semua tools sudah berjalan

2.3 Konsep Kunci

Git vs GitHub CLI: Apa Bedanya?

Sebelum install, penting memahami perbedaan dua tools ini karena keduanya akan kita gunakan (lihat Tabel 2.1).

Git (git)

Sistem version control yang berjalan di komputer lokal Anda. Digunakan untuk:

- Operasi version control: `commit`, `branch`, `merge`, `diff`, `log`
- Operasi basic remote: `push`, `pull`, `fetch`, `clone`
- Semua yang berhubungan dengan melacak perubahan kode

GitHub CLI (gh)

Tools khusus untuk berinteraksi dengan GitHub. Digunakan untuk:

- Membuat repository di GitHub: `gh repo create`
- Mengelola Pull Request: `gh pr create`, `gh pr view`, `gh pr merge`
- Mengelola Issues: `gh issue create`, `gh issue list`
- Workflow dan automation GitHub lainnya

Analogi: Git seperti Microsoft Word (untuk menulis dokumen), GitHub CLI seperti plugin yang menghubungkan Word dengan Google Drive (untuk upload dan kolaborasi online).

Yang perlu diingat: `git push` dan `git pull` masih dipakai normal. GitHub CLI hanya menambahkan kemampuan untuk operasi yang lebih spesifik ke GitHub.

Kapan pakai Git vs GitHub CLI:

- **Git (Lokal):** Version control lokal - `commit`, `branch`, `merge`, `log`, `diff`
- **Sinkronisasi:** Hubungkan lokal remote - `push`, `pull`, `clone`, `fetch`, `remote`
- **GitHub CLI (gh):** Fitur GitHub spesifik - PR, issues, workflows, repo management

Line Ending: CRLF vs LF

Ini topik yang sering bikin panik pemula, padahal sebenarnya sederhana. Windows menggunakan CRLF (Carriage Return +

Git (Lokal)	Sinkronisasi	GitHub CLI (Fitur GitHub)
git commit	git push	gh pr create
git branch	git pull	gh pr merge
git merge	git clone	gh pr review
git log	git fetch	gh repo create
git diff	git remote	gh issue list
git status	—	gh workflow run

Tabel 2.1: *Tabel perbandingan Git vs GitHub CLI*

Line Feed) untuk akhir baris. Linux/Mac menggunakan LF saja. Kalau Anda kerja dengan tim yang pakai OS berbeda, ini bisa jadi masalah kecil.

Git punya fitur `autocrlf` yang otomatis konversi:

- Saat Anda commit: CRLF → LF (standar universal)
- Saat Anda checkout: LF → CRLF (sesuai Windows)

Kita akan setting `autocrlf=true` saat install. Dengan begitu, Anda tidak perlu pusing soal ini.

2.4 Praktik CLI: Install dan Konfigurasi

1. Install Git for Windows

- (a) Buka browser, dan download installer di <https://git-scm.com/download/win>
- (b) Jalankan installer dengan opsi penting:
 - **Default editor:** Pilih VS Code (jika sudah terinstall)
 - **PATH environment:** "Git from the command line and also from 3rd-party software"
 - **Line ending:** "Checkout Windows-style, commit Unix-

style" (autocrlf=true)

- **Terminal emulator:** "Use MinTTY"
- Opsi lain: biarkan default saja

2. Verifikasi Git dengan mengetik di Git Bash.

```
$ git --version
```

Output:

```
git version 2.42.0.windows.1
```

Versi yang ditampilkan mungkin berbeda versi, tetapi tidak masalah. Yang penting command berhasil berjalan tanpa error.

3. Install VS Code

- (a) Kunjungi <https://code.visualstudio.com/> dan klik tombol download
- (b) Jalankan installer
- (c) Centang opsi "Add to PATH" (penting agar bisa buka VS Code dari terminal). Install hingga selesai.

4. Install GitHub CLI

- (a) Kunjungi <https://cli.github.com/> dan download installer untuk Windows
- (b) Jalankan installer, dan ikuti wizard menggunakan default settings

5. Verifikasi GitHub CLI

Tutup dan buka ulang Git Bash (agar PATH ter-refresh), lalu ketik:

```
$ gh --version
```

Output:

```
gh version 2.40.0 (2023-12-05)
```

6. Konfigurasi Identitas Git

Git perlu tahu siapa Anda agar setiap commit tercatat dengan nama dan email Anda.

```
$ git config --global user.name "Nama Anda"
$ git config --global user.email "email@example.com"
```

Penting

Gunakan email yang digunakan di GitHub. Untuk menghindari kesalahan, jangan meng-*copy* langsung perintah di atas!

7. Set Editor Default (Opsional)

Kalau Anda ingin Git membuka VS Code saat dibutuhkan edit pesan, salin perintah berikut:

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

8. Verifikasi Konfigurasi

Lihat semua konfigurasi Git Anda:

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

Contoh Output (sebagian):

```
user.name=Dian Hanifudin Subhi
user.email=dhanifudin@gmail.com
core.editor=code --wait
core.autocrlf=true
```

Pastikan `user.name` dan `user.email` sudah benar. Ulangi langkah sebelumnya jika hasil luaran tidak sesuai dengan identitas yang Anda inginkan.

9. Autentikasi GitHub CLI

Sekarang kita hubungkan GitHub CLI dengan akun GitHub Anda.

```
$ gh auth login
```

Anda akan melihat pertanyaan interaktif:

```
? What account do you want to log into?  
> GitHub.com  
    GitHub Enterprise Server
```

Pilih **GitHub.com** (pakai arrow keys, tekan Enter).

```
? What is your preferred protocol for Git operations?  
> HTTPS  
    SSH
```

Pilih **HTTPS** (lebih mudah untuk pemula).

```
? Authenticate Git with your GitHub credentials?  
> Yes  
    No
```

Pilih **Yes**.

```
? How would you like to authenticate GitHub CLI?  
> Login with a web browser  
    Paste an authentication token
```

Pilih **Login with a web browser**.

Anda akan diberi kode seperti `XXXX-XXXX`. Tekan Enter, dan browser akan terbuka. Login ke GitHub, paste kode tersebut,

dan authorize GitHub CLI. Kalau berhasil, Anda akan lihat pesan berikut (username yang ditampilkan sesuai dengan akun GitHub Anda):

```
Logged in as dhanifudin
```

10. Verifikasi Autentikasi

```
$ gh auth status
```

Output:

```
github.com
  Logged in to github.com as dhanifudin (oauth_token)
  Git operations for github.com configured to use https
  protocol.
```

Kalau ada centang hijau, berarti setup berhasil!

Praktik VS Code: Integrasi dengan Git

1. Launch VS Code dari Start Menu atau ketik code di Git Bash.
2. Install Extension Git Graph (Recommended)
 - (a) Pada VS Code buka Extensions (**Ctrl+Shift+X**)
 - (b) Cari "Git Graph", Install
3. Cek Integrasi Git
 - (a) Buka terminal di VS Code **Ctrl+`** (backtick)
 - (b) Pastikan dropdown terminal = "Git Bash"
 - (c) Ketik `git --version`
4. Buka Folder 'web-tanpa-drama'
 - (a) **File → Open Folder**
 - (b) Pilih folder `web-tanpa-drama` (dari Bab 1)

Panel Source Control masih kosong. Itu normal, karena belum

dilakukan inisialisasi repository Git.

Kesalahan Umum

1. **Lupa restart terminal setelah install** Setelah install Git atau GitHub CLI, PATH baru tidak langsung dikenali. Tutup dan buka ulang terminal. Kalau masih tidak bisa, restart komputer.
2. **Install Git tapi tidak set identitas** Anda bisa mulai pakai Git tanpa set `user.name` dan `user.email`, tapi nanti saat commit pertama akan error. Lebih baik set sejak awal.
3. **Bingung bedanya Git dan GitHub CLI** Git untuk version control, GitHub CLI untuk operasi GitHub. Anda butuh keduanya. `git` dipakai untuk commit lokal, `gh` dipakai untuk buat PR dan manage repo GitHub.
4. **Autentikasi GitHub CLI gagal** Kalau `gh auth login` gagal, pastikan:
 - (a) Koneksi internet stabil
 - (b) Browser tidak block pop-up
 - (c) Akun GitHub sudah terdaftar
 - (d) Kalau masih gagal, coba lagi dengan pilihan "Paste an authentication token" (buat token manual di GitHub settings).
5. **VS Code tidak mengenali Git** Kalau panel Source Control di VS Code tidak berfungsi, pastikan Git sudah terinstall dan `git --version` jalan di terminal. Restart VS Code.
6. **Pakai email yang berbeda di Git dan GitHub** Kalau email di `git config` beda dengan email akun GitHub, commit Anda tidak akan muncul di profil GitHub. Pastikan menggunakan email yang sama.

2.5 Latihan

Instruksi

Jalankan command berikut di Git Bash dan amati hasilnya:

```
$ git --version
$ gh --version
$ git config --global user.name
$ git config --global user.email
$ gh auth status
$ code --version
```

Hasil yang Diharapkan

Semua command memberikan output tanpa error:

- x Git version muncul
- x GitHub CLI version muncul
- x User name sudah terset
- x User email sudah terset
- x Autentikasi GitHub berhasil (logged in)
- x VS Code version muncul

2.6 Ringkasan

Setup lengkap untuk belajar Git di Windows meliputi:

1. **Git**: Sistem version control lokal (commit, branch, merge, push, pull)
2. **VS Code**: Editor kode yang terintegrasi dengan Git
3. **GitHub CLI**: Tools untuk operasi GitHub seperti PR, issue dan repo management
Konfigurasi wajib:
 - `git config --global user.name`

- `git config --global user.email`
- `gh auth login` untuk autentikasi GitHub

Di bab berikutnya, kita akan mulai menggunakan Git sungguhan: inisialisasi repository pertama!

2.7 Cheat Commands

```
# Verifikasi instalasi
$ git --version
$ gh --version
$ code --version

# Konfigurasi identitas
$ git config --global user.name "Nama Anda"
$ git config --global user.email "email@example.com"
$ git config --global core.editor "code --wait"

# Lihat konfigurasi
$ git config --list
$ git config user.name
$ git config user.email

# Autentikasi GitHub
$ gh auth login
$ gh auth status
```

Sekarang setup Anda sudah siap. Saatnya mulai praktik Git yang sesungguhnya!

3. Repo Pertama: `init` & `status`

3.1 Kenapa Bab Ini Penting

Anda akan membuat repository Git pertama dan belajar menggunakan `git status` command yang akan menjadi sahabat terbaik. Command `git status` adalah kompas yang memberi tahu posisi Anda: file mana yang berubah, mana yang sudah di-*stage*, mana yang belum. Tanpa ini, Anda akan tersesat. Bab ini adalah fondasi praktikal sebelum kita masuk ke `commit` dan operasi lain.

3.2 Tujuan Belajar

- Menginisialisasi folder sebagai Git repository dengan `git init`
- Memahami arti output dari `git status`
- Membedakan file "untracked", "modified", dan "staged"
- Menggunakan VS Code Source Control untuk melihat perubahan
- Memahami struktur folder `.git` secara konseptual

3.3 Konsep Kunci

Repository: Folder dengan Kekuatan Super

Sebuah folder biasa menjadi **Git repository** setelah `git init`. Git membuat folder tersembunyi `.git` yang menyimpan semua riwayat dan metadata. Jangan edit manual folder `.git`.

Status: Tiga Kondisi File

File dalam repository Git bisa berada di tiga kondisi:

1. **Untracked** (Tidak dilacak)

File baru yang belum pernah di-commit. Git tahu file ini ada, tapi tidak akan merekam perubahannya sampai Anda bilang (via `git add`).

Analogi: Kertas yang baru Anda letakkan di meja. Git tahu ada kertas baru, tapi belum tahu mau diapakan.

2. **Modified** (Sudah diubah tapi belum di-stage)

File yang sudah pernah di-commit sebelumnya, lalu Anda edit. Perubahan belum masuk staging area.

Analogi: Kertas yang sudah ada arsipnya, Anda coret-core, tapi belum Anda masukkan ke keranjang "akan diarsipkan".

3. **Staged** (Siap di-commit)

File yang sudah masuk staging area via `git add`. Perubahan ini siap untuk disimpan permanen dengan `git commit`.

Analogi: Kertas yang sudah Anda masukkan ke keranjang "akan diarsipkan". Tinggal pencet tombol arsip (commit), dan tersimpan permanen. Ilustrasi alur perubahan file dalam Git ada di Gambar 3.1.



Gambar 3.1: Diagram alur perubahan file dalam Git

Alur perubahan: *Working Directory (untracked/modified)* → *git add* → *Staging Area (staged)* → *git commit* → *Repository (committed)*. Gunakan *git status* untuk cek posisi file.

Git Status: Kompas Anda

`git status` adalah command paling sering dipakai. Fungsinya:

- Branch mana yang aktif
- File apa yang berubah
- File mana yang sudah di-stage
- File mana yang untracked

Kalau Anda bingung, ketik `git status`. Maka sebagian kebingungan bisa terjawab dari sini.

3.4 Praktik CLI: Inisialisasi dan Status

1. Masuk ke Folder Proyek

Buka Git Bash, masuk ke folder `web-tanpa-drama` yang sudah kita buat sebelumnya:

```
$ cd ~/web-tanpa-drama
```

2. Inisialisasi Repository

Sekarang kita ubah folder biasa ini menjadi Git repository:

```
$ git init
```

Output:

```
Initialized empty Git repository in
C:/Users/dian/web-tanpa-drama/.git/
```

Selamat! Folder ini sekarang adalah repository Git. Folder `.git` sudah dibuat (tersembunyi, tapi ada).

3. Cek Status Pertama Kali

```
$ git status
```

Output:

```
On branch main
No commits yet
```

```
Untracked files:
```

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

```
    README.md
```

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

Penjelasan:

- **On branch main:** Branch default
- **No commits yet:** Belum ada commit
- **Untracked files:** README.md belum dilacak

Git memberi hint: gunakan `git add` untuk mulai melacak file ini. Tapi belum! Kita eksplorasi dulu.

4. Buat File HTML Sederhana

Buat file `index.html` langsung dari terminal Git Bash:

```
$ cat > index.html << 'EOF'
<!DOCTYPE html>
<html>
<head>
  <title>Web Tanpa Drama</title>
</head>
<body>
  <h1>Selamat Datang di Web Tanpa Drama</h1>
  <p>Ini adalah proyek latihan Git.</p>
</body>
</html>
EOF
```

5. Cek Status Lagi

```
$ git status
```

Output:

```
On branch main
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be
    committed)
        README.md
        index.html

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

Sekarang ada dua file untracked: README.md dan index.html. Git tahu ada file baru, tapi belum melacak perubahan karena belum kita add.

6. Lihat Isi Folder (Termasuk Hidden)

```
$ ls -la
```

Output:

```
total 8
drwxr-xr-x 1 dian 197121  0 Jan 27 10:15 ./
drwxr-xr-x 1 dian 197121  0 Jan 27 09:30 ../
drwxr-xr-x 1 dian 197121  0 Jan 27 10:15 .git/
-rw-r--r-- 1 dian 197121 123 Jan 27 10:16 index.html
-rw-r--r-- 1 dian 197121 18 Jan 27 09:35 README.md
```

Lihat terdapat folder `.git/` di sana! Itu adalah "database" repository Anda. Jangan hapus atau edit manual folder ini.

7. Edit File yang Sudah Ada

```
$ echo "" >> README.md
$ echo "Proyek latihan Git untuk pemula." >> README.md
```

Sekarang kita edit README.md untuk melihat perbedaan antara "untracked" dan "modified".

8. Cek Status Sekali Lagi

```
$ git status
```

Tunggu, kenapa README.md masih untracked, bukan modified?

Jawaban: Karena README.md belum pernah di-commit sama sekali. File yang "modified" adalah file yang **sudah pernah di-commit sebelumnya lalu diubah**. Karena belum ada commit pertama, semua file masih dianggap untracked.

Konsep ini akan jelas nanti setelah kita commit pertama kali di Bab 4.

3.5 Praktik VS Code: Source Control Panel

1. Buka Folder di VS Code

- (a) Buka VS Code
- (b) **File → Open Folder**
- (c) Pilih web-tanpa-drama

2. Lihat Panel Source Control

Klik ikon Git di sidebar kiri (Ctrl+Shift+G).

Panel Source Control aktif:

- **Changes:** File README.md dan index.html dengan label **U** (Untracked)

3. Bedakan "Changes" dan "Staged Changes"

- **Staged Changes:** File siap commit (kosong sekarang)
- **Changes:** File untracked atau modified belum di-stage

4. Lihat Diff

Klik file README.md di panel Changes. VS Code buka diff

viewer:

- Kiri: kosong (belum ada commit)
- Kanan: isi sekarang
- Garis hijau: baris ditambah

3.6 Kesalahan Umum

1. **Lupa git init** Langsung ketik `git status` di folder biasa (bukan repository). Error: "not a git repository". Solusi: Jalankan `git init` dulu.
2. **Bingung kenapa file tidak berubah setelah edit** Anda edit file, tapi `git status` tidak berubah. Solusi: Pastikan Anda save file tersebut (di editor atau terminal). Git hanya tahu perubahan yang sudah tersimpan ke disk.
3. **Panik melihat "untracked files"** Output `git status` penuh dengan file untracked, lalu panik "apakah ini masalah?" Solusi: Tenang. Untracked files normal. Anda yang tentukan mana yang mau di-track dengan `git add`.
4. **Hapus folder .git secara tidak sengaja** Folder `.git` terhapus, semua riwayat hilang. Repository jadi folder biasa lagi. Solusi: Jangan hapus folder `.git`. Kalau mau "uninstall" Git dari folder, baru hapus `.git` (tapi Anda akan kehilangan semua riwayat).
5. **Mengira git status mengubah sesuatu** Anda takut ketik `git status` karena pikir akan mengubah file. Solusi: `git status` hanya **membaca** kondisi. Tidak mengubah apapun. Aman dipakai kapan saja.
6. **Tidak pakai git status sama sekali** Langsung ketik `commit` lain tanpa cek kondisi dulu. Solusi: Biasakan selalu

ketik `git status` sebelum dan sesudah command penting. Ini kebiasaan baik yang akan menghindarkan Anda dari banyak masalah.

3.7 Latihan

Instruksi

1. Buat file baru `about.html` (copy dari `index.html` lalu ubah sedikit)
2. Edit `README.md` sekali lagi, tambahkan satu baris apa saja
3. Jalankan `git status` dan amati output
4. Cek di VS Code Source Control, lihat berapa file di "Changes"

Hasil yang Diharapkan

`git status` menampilkan:

- **Untracked files:** `README.md`, `index.html`, `about.html` (total 3 file)

Di VS Code: 3 file di "Changes" dengan label **U**.

3.8 Ringkasan

- **git init** mengubah folder biasa menjadi Git repository dengan membuat folder `.git`.
- **git status** adalah kompas Anda: menunjukkan kondisi file saat ini (untracked, modified, staged).
- File punya tiga kondisi utama:
 - **Untracked:** Belum pernah di-commit
 - **Modified:** Sudah pernah di-commit, lalu diubah
 - **Staged:** Siap di-commit
- Folder `.git` menyimpan semua riwayat dan metadata. Jangan edit atau hapus folder ini.

- VS Code Source Control panel memvisualisasi kondisi file secara visual. "Changes" untuk file belum di-stage, "Staged Changes" untuk file siap commit.

Di bab berikutnya, kita akan belajar cara **stage** file dengan `git add` dan **commit** pertama kali!

3.9 Cheat Commands

```
# Inisialisasi folder sebagai Git repository
$ git init
# Lihat kondisi repository (wajib sering dipakai!)
$ git status
# Lihat isi folder termasuk hidden files
$ ls -la
# Buat file dengan isi multiline (EOF sebagai delimiter)
$ cat > file.txt << 'EOF'
```

Selamat! Anda sudah berhasil membuat repository pertama dan memahami cara membaca `git status`. Sekarang saatnya mempelajari salah satu perintah paling penting dalam Git: **commit**.

Di bab berikutnya, Anda akan belajar cara menyimpan snapshot perubahan dengan rapi, menulis pesan commit yang bermakna, dan memahami workflow `git add` → `git commit`. Setiap commit adalah checkpoint yang akan menyelamatkan Anda dari drama kehilangan kode atau perubahan yang salah. Mari lanjut ke Bab 4 untuk menguasai seni commit yang profesional!

4. Commit yang Rapi: add & commit

4.1 Kenapa Bab Ini Penting

Commit adalah jantung dari Git. Setiap commit adalah snapshot permanen dari proyek Anda pada satu titik waktu. Kalau Anda commit dengan baik, memilih perubahan yang tepat, menulis pesan yang jelas, riwayat proyek Anda akan rapi dan mudah dipahami. Kalau commit asal-asalan, riwayat jadi berantakan dan tidak membantu. Bab ini mengajarkan cara commit yang profesional sejak awal.

4.2 Tujuan Belajar

- Menggunakan `git add` untuk memilih file yang akan di-commit
- Membuat commit pertama dengan `git commit`
- Menulis pesan commit yang jelas dan bermanfaat
- Memahami perbedaan `git add <file>` dan `git add .` (titik)
- Mengerti konvensi pesan commit (feat, fix, docs)
- Melakukan commit via VS Code Source Control

4.3 Konsep Kunci

Staging Area: Keranjang Belanja

Sebelum Anda commit, Anda perlu memilih file mana yang akan masuk ke commit tersebut. Inilah fungsi **staging area** tempat Anda menaruh file yang siap di-commit.

Analogi: Keranjang belanja di supermarket. Anda berjalan-jalan, ambil barang, masukkan ke keranjang. Barang yang di keranjang akan dibeli (di-commit). Barang di rak lain tidak ikut

checkout.

Kenapa perlu staging? Supaya Anda bisa bikin commit yang **fokus**. Misalnya, Anda ubah 5 file sekaligus, tapi hanya 3 file yang berhubungan dengan satu fitur. Anda stage 3 file itu, commit dengan pesan yang menjelaskan fitur tersebut. 2 file lainnya di-commit terpisah dengan pesan berbeda.

Git Add: Dua Cara

1. Add file spesifik

```
git add namafile.txt
```

Hanya `namafile.txt` yang masuk staging.

2. Add semua perubahan

```
git add .
```

Semua file yang berubah (untracked, modified) masuk staging sekaligus.

Kapan pakai yang mana?

- Pakai `git add <file>` kalau Anda ingin commit bertahap (lebih rapi, lebih profesional).
- Pakai `git add .` kalau semua perubahan memang berhubungan dan siap di-commit sekaligus.

Risiko `git add .`

Anda mungkin tanpa sadar commit file yang tidak perlu (misalnya file konfigurasi lokal, file temporary). Hati-hati.

Pesan commit yang baik menjelaskan **APA** yang diubah dan **KENAPA**. Hindari pesan seperti:

4. Commit yang Rapi: *add & commit*

- "update"
- "fix"
- "changes"
- "asdf"

Pesan yang buruk ini tidak membantu ketika 3 bulan kemudian Anda atau teman Anda mencari commit yang memperbaiki bug tertentu.

Contoh pesan yang baik:

- "Add login form to homepage"
- "Fix broken navigation link on about page"
- "Update README with installation instructions"

Rekomendasi: Gunakan bahasa Inggris. Alasannya:

- Standar industri
- Kalau proyek open-source, orang internasional bisa paham
- Pesan commit tidak perlu sempurna grammar-nya, cukup jelas

Untuk proyek kecil, pesan bebas asal jelas. Untuk proyek kompleks atau tim, pakai konvensi seperti:

Konvensi Conventional Commits (ringan):

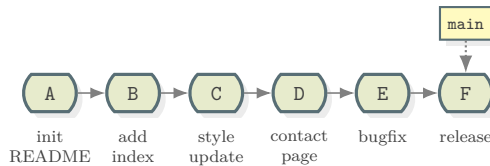
- **feat:** Fitur baru (feature)
- **fix:** Perbaikan bug
- **docs:** Perubahan dokumentasi
- **style:** Format, spacing (tidak mengubah logika)
- **refactor:** Refactor kode (tidak menambah fitur atau fix bug)

Contoh (visualisasinya lihat Gambar 4.1):

- feat: add contact form to homepage
- fix: resolve broken image link
- docs: update README installation steps

Kita akan pakai konvensi ini untuk latihan yang agak kompleks.

Referensi: Konvensi ini mengikuti spesifikasi Conventional Commits (<https://www.conventionalcommits.org/>), standar industri untuk pesan commit yang konsisten.



Gambar 4.1: *Timeline commit dengan pesan singkat*

4.4 Praktik CLI: Commit Pertama

1. Pastikan di Folder yang Benar

```
$ cd ~/web-tanpa-drama
```

2. Cek Status

Selalu mulai dengan `git status`:

```
$ git status
```

Output (seharusnya):

```
On branch main

No commits yet

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

```
README.md
about.html
index.html
```

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

Ada 3 file untracked (dari latihan Bab 3).

3. Stage File README.md

Kita mulai dengan commit yang fokus: dokumentasi dulu.

```
$ git add README.md
```

Tidak ada output, itu normal. Kalau tidak ada error, berarti berhasil.

4. Cek Status Lagi

```
$ git status
```

Output:

```
On branch main

No commits yet

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

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

Lihat perbedaannya? `README.md` sekarang di section **"Changes to be committed"** (staged). File lain masih untracked.

5. Commit Pertama!

Sekarang kita commit `README.md` dengan pesan yang jelas:

```
$ git commit -m "docs: add initial README"
```

Output:

```
[main (root-commit) a3f8b2c] docs: add initial README
1 file changed, 3 insertions(+)
create mode 100644 README.md
```

Penjelasan:

- `[main ...]`: Commit berhasil di branch main
 - `(root-commit)`: Ini commit pertama (root)
 - `a3f8b2c`: Hash pendek dari commit ini
 - `docs: add initial README`: Pesan commit Anda
 - `1 file changed, 3 insertions(+)`: Statistik perubahan
- Selamat, Anda baru saja membuat commit pertama!

6. Cek Status Setelah Commit

```
$ git status
```

Output:

```
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be
   committed)
    about.html
    index.html
```

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

README.md sudah tidak muncul lagi karena sudah di-commit. Yang tersisa hanya file untracked.

7. Edit File HTML

Sekarang kita siapkan `index.html` untuk commit kedua. Buka dan edit file:

```
$ cat > index.html << 'EOF'
<!DOCTYPE html>
<html lang="id">
<head>
  <meta charset="UTF-8">
  <title>Web Tanpa Drama</title>
</head>
<body>
  <h1>Selamat Datang di Web Tanpa Drama</h1>
  <p>Ini adalah proyek latihan Git untuk pemula.</p>
  <p>Belajar Git tidak harus drama!</p>
</body>
</html>
EOF
```

Kita ubah isi `index.html` dengan versi yang lebih lengkap.

8. Stage dan Commit index.html

```
$ git add index.html
$ git commit -m "feat: add basic homepage structure"
```

Output:

```
[main b7e4d1a] feat: add basic homepage structure
1 file changed, 10 insertions(+)
```

```
create mode 100644 index.html
```

9. Edit dan Commit about.html

Sekarang giliran `about.html`. Buat isinya:

```
$ cat > about.html << 'EOF'
<!DOCTYPE html>
<html lang="id">
<head>
  <meta charset="UTF-8">
  <title>Tentang - Web Tanpa Drama</title>
</head>
<body>
  <h1>Tentang Proyek Ini</h1>
  <p>Web Tanpa Drama adalah proyek latihan untuk belajar
    Git dari nol.</p>
  <p>Setiap file di sini adalah bagian dari latihan
    praktis.</p>
</body>
</html>
EOF
```

Stage dan commit:

```
$ git add about.html
$ git commit -m "feat: add about page"
```

Output:

```
[main c9a5e3f] feat: add about page
1 file changed, 10 insertions(+)
create mode 100644 about.html
```

10. Cek Status Terakhir

```
$ git status
```

Output:

```
On branch main
nothing to commit, working tree clean
```

”nothing to commit, working tree clean” adalah output yang indah! Artinya semua perubahan sudah di-commit, tidak ada yang tertinggal.

4.5 Praktik VS Code: Commit via GUI

Sekarang kita coba commit menggunakan VS Code Source Control.

1. Edit README.md

Buka README.md di VS Code, tambahkan baris baru:

```
# Web Tanpa Drama

Proyek latihan Git untuk pemula.

## Tujuan
Belajar Git dari nol hingga mahir berkolaborasi di GitHub.
```

Save file README.md tersebut (Ctrl+S).

2. Lihat Perubahan di Source Control

Klik ikon Git di sidebar kiri. Anda akan melihat README.md muncul di **”Changes”** dengan label **M** (Modified).

Klik file README.md untuk melihat diff:

- Baris yang ditambahkan berwarna hijau
- Baris yang dihapus (kalau ada) berwarna merah

3. Stage File

Hover mouse di atas README.md di panel Changes. Akan muncul ikon **”+”**. Klik ikon tersebut.

4. Commit yang Rapi: add & commit

README.md akan pindah ke section **"Staged Changes"**.

Alternatif: Klik tanda "+" di sebelah tulisan "Changes" untuk stage semua file sekaligus.

4. Tulis Pesan Commit

Di bagian atas panel Source Control, ada text box dengan placeholder "Message". Klik di sana dan ketik:

```
docs: add project goals to README
```

5. Commit

Klik tombol **Commit** (biasanya ada ikon atau tulisan "Commit").

Setelah commit, README.md hilang dari panel Source Control. Working tree clean lagi.

6. Verifikasi di Terminal

Buka terminal di VS Code (' Ctrl+ `), lalu ketik:

```
$ git status
```

Output:

```
On branch main
nothing to commit, working tree clean
```

Commit berhasil!

4.6 Kesalahan Umum

1. **Lupa stage sebelum commit** Error: "nothing to commit". Anda langsung git commit tanpa git add. Solusi: Stage file dulu dengan git add.
2. **Commit terlalu banyak perubahan sekaligus** Commit berisi 10 file dengan perubahan berbeda-beda, pesan commit

"update". Solusi: Commit sedikit demi sedikit dengan pesan yang spesifik. Lebih baik banyak commit kecil daripada satu commit besar.

3. **Pesan commit tidak jelas** Contoh buruk: "update", "fix", "changes". Solusi: Jelaskan APA dan KENAPA. "Fix broken link on about page" jauh lebih baik.
4. **Pakai git add . tanpa cek** File yang tidak perlu ikut ter-commit (misalnya file temporary, konfigurasi lokal). Solusi: Pakai `git status` sebelum `git add .` untuk memastikan tidak ada file yang tidak diinginkan.
5. **Commit file yang belum di-save** Anda edit di editor, tapi lupa save. `git status` tidak menunjukkan perubahan. Solusi: Selalu save file sebelum stage dan commit.
6. **Takut commit karena "belum sempurna"** Menunda commit sampai fitur benar-benar selesai dan polish. Solusi: Commit boleh kecil dan sering. "Work in progress" juga boleh di-commit selama pesannya jelas (misalnya "feat: add login form (WIP)").

4.7 Latihan

Instruksi

Buat 3 commit kecil dengan pesan yang jelas. Ikuti skenario ini:

1. Buat file `style.css` dengan isi CSS sederhana (misal: `body font-family: Arial;`). Commit dengan pesan `feat: add basic stylesheet`.
2. Edit `index.html`, tambahkan link ke `style.css` di bagian `<head>`:

```
<link rel="stylesheet" href="style.css">
```

- Commit dengan pesan `feat: link stylesheet to homepage`.
3. Edit `about.html`, tambahkan juga link ke `style.css`. Commit dengan pesan `feat: link stylesheet to about page`.

Hasil yang Diharapkan

Setelah selesai, `git status` harus menunjukkan "nothing to commit, working tree clean". Anda seharusnya punya 3 commit baru dengan pesan yang jelas dan spesifik. Nanti di Bab 5, kita akan lihat riwayat commit ini dengan `git log`.

4.8 Ringkasan

Proses commit:

1. Edit file (working directory)
2. Stage dengan `git add` (staging area)
3. Commit dengan `git commit -m "message"` (repository)

Perintah penting:

- `git add <file>`: Stage file tertentu
- `git add .`: Stage semua perubahan
- `git commit -m "message"`: Commit dengan pesan singkat

Konvensi pesan commit:

- **feat**: Fitur baru
- **fix**: Perbaikan bug
- **docs**: Dokumentasi
- Gunakan bahasa Inggris, jelas, ringkas

Prinsip commit yang baik:

- Commit sering, jangan tunggu sempurna
- Satu commit = satu fokus perubahan
- Pesan commit menjelaskan APA dan KENAPA

- Selalu cek `git status` sebelum dan sesudah commit

Di bab berikutnya, kita akan belajar cara melihat riwayat commit dengan `git log` dan melihat perbedaan antar-versi dengan `git diff`.

4.9 Cheat Commands

```
$ git add <file>           # Stage file tertentu
$ git add .                 # Stage semua perubahan (titik)
$ git commit -m "message"  # Commit dengan pesan
$ git status                # Cek kondisi repository (selalu!)
```

Selamat! Anda sudah bisa membuat commit yang rapi. Riwayat proyek Anda sekarang tercatat dengan baik.

5. Membaca Perubahan: log & diff

5.1 Kenapa Bab Ini Penting

Setelah Anda mulai commit, proyek Anda punya riwayat. Tapi riwayat tidak berguna kalau Anda tidak tahu cara membacanya. `git log` membantu Anda melihat commit apa saja yang sudah dibuat, siapa yang buat, dan kapan. `git diff` membantu Anda melihat perubahan apa yang terjadi, sebelum commit, setelah commit, atau antar dua commit. Bab ini mengajarkan cara menjelajahi riwayat proyek Anda dengan percaya diri.

5.2 Tujuan Belajar

- Melihat riwayat commit dengan `git log`
- Menggunakan `git log --oneline --graph` untuk visualisasi ringkas
- Melihat perubahan belum/sudah di-stage dengan `git diff`
- Mencari commit yang menyebabkan bug sederhana
- Menggunakan diff viewer di VS Code

5.3 Konsep Kunci

Git Log: Riwayat Lengkap

`git log` menampilkan daftar commit (terbaru → terlama). Setiap commit punya:

- **Hash (SHA):** ID unik
- **Author:** Nama dan email
- **Date:** Kapan dibuat

- **Message:** Pesan commit

Secara default, `git log` menampilkan output yang panjang dan detail. Untuk penggunaan sehari-hari, kita akan pakai versi yang lebih ringkas: `git log --oneline --graph`.

Git Diff: Melihat Perbedaan

`git diff` membandingkan versi file. Ada beberapa varian:

1. **`git diff` (tanpa parameter)** Menampilkan perubahan di working directory yang belum di-stage. Alias: "apa yang berubah sejak commit terakhir tapi belum di-add?"
2. **`git diff --staged` (atau `git diff --cached`)** Menampilkan perubahan yang sudah di-stage. Alias: "apa yang akan masuk ke commit berikutnya?"
3. **`git diff <commit>` (advanced, tidak dibahas di buku ini)** Membandingkan working directory dengan commit tertentu.

Output diff menggunakan format standar:

- **Baris hijau (+):** Ditambahkan
- **Baris merah (-):** Dihapus
- **Baris putih:** Tidak berubah (konteks)

Kenapa Ini Penting?

Bayangkan Anda kerja seharian, edit banyak file. Sebelum commit, Anda lupa apa saja yang diubah. Tanpa `git diff`, Anda bisa saja commit perubahan yang salah atau tidak sengaja. Dengan `git diff`, Anda bisa review dulu sebelum commit, mencegah penyesalan.

Begitu juga dengan `git log`. Kalau ada bug muncul tiba-tiba, Anda bisa lihat commit terakhir: "Ah, kemarin saya ubah fungsi ini, mungkin itu penyebabnya."

5.4 Praktik CLI: Melihat Riwayat dan Perubahan

1. Melihat Log

```
$ cd ~/web-tanpa-drama
$ git log
```

Output ini panjang dan detail. Kalau Anda punya banyak commit, akan sulit dibaca. Tekan `q` untuk keluar dari log viewer (kalau tidak muat di satu layar, Git menggunakan pager untuk menampilkan log).

Untuk menampilkan log ringkas tambahkan opsi `--oneline`:

```
$ git log --oneline
```

Output mungkin akan berbeda:

```
c9a5e3f feat: add about page
b7e4d1a feat: add basic homepage structure
a3f8b2c docs: add initial README
```

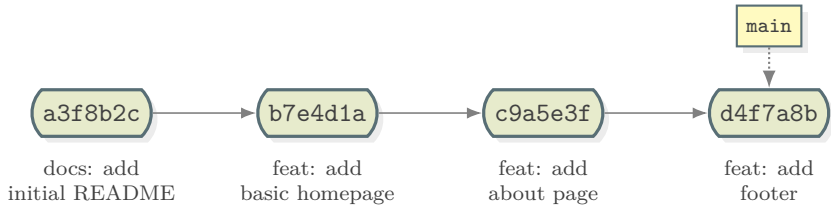
2. Log dengan Grafik

```
$ git log --oneline --graph
```

Output (fitur ini baru akan berguna ketika ada branching):

```
* c9a5e3f feat: add about page
* b7e4d1a feat: add basic homepage structure
* a3f8b2c docs: add initial README
```

Diagram timeline commit pada 5.1 menampilkan alur perubahan dari kiri ke kanan, masing-masing berlabel dengan hash pendek dan pesan commit. Ini menunjukkan riwayat linear proyek Anda.



Gambar 5.1: *Timeline commit*

3. Edit File dan Lihat Diff

Edit `index.html`, tambahkan footer:

```
$ cat > index.html << 'EOF'
<!DOCTYPE html>
<html lang="id">
<head>
  <meta charset="UTF-8">
  <title>Web Tanpa Drama</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>Selamat Datang di Web Tanpa Drama</h1>
  <p>Belajar Git tidak harus drama!</p>
  <footer>
    <p>&copy; 2025 Web Tanpa Drama</p>
  </footer>
</body>
</html>
EOF
```

Cek diff:

```
$ git diff
```

Output (contoh):

```
+ <footer>
+ <p>&copy; 2025 Web Tanpa Drama</p>
+ </footer>
```

4. Stage dan Lihat Diff Staged

```
$ git add index.html
$ git diff
```

Output **kosong**, perubahan sudah di-stage.

Lihat yang di-stage:

```
$ git diff --staged
```

Commit:

```
$ git commit -m "feat: add footer to homepage"
```

5. Melihat Log dan Debug

Log lagi:

```
$ git log --oneline --graph
```

Lihat detail commit tertentu, commit hash bisa diambil dari output log (misal d4f7a8b):

```
$ git show d4f7a8b
```

5.5 Praktik VS Code: Diff Viewer

1. Edit dan Lihat Diff

Edit `about.html`, tambahkan footer. Buka Source Control, klik `about.html`.

VS Code tampilkan **diff viewer** dua panel:

- Kiri: versi lama

- Kanan: versi baru
2. Stage dan Commit
Stage dengan "+", commit:

```
feat: add footer to about page
```

Timeline (Opsional): Klik kanan file → "Open Timeline" untuk lihat history file.

5.6 Kesalahan Umum

- (a) **Lupa cek diff sebelum commit** → Bisa commit kode debug tidak sengaja.
- (b) **git diff kosong padahal sudah edit** → Lupa save atau sudah di-stage (pakai `git diff --staged`).
- (c) **Stuck di log pager** → Tekan q.

5.7 Latihan

1. Tambahkan section "## Teknologi" di README.md.
2. **Sebelum stage:** `git diff` → lihat perubahan.
3. Stage: `git add README.md`.
4. `git diff --staged` → lihat yang di-stage.
5. Commit: docs: add technology section to README.
6. `git log --oneline --graph` → commit baru di atas.

5.8 Ringkasan

Git Log:

- `git log --oneline`: ringkas (1 baris/commit)
- `git log --oneline --graph`: dengan visualisasi
- `git show <commit>`: detail satu commit

Git Diff:

- `git diff`: perubahan belum di-stage
- `git diff --staged`: perubahan sudah di-stage
- Output: + = ditambah, - = dihapus

Best Practices:

- Cek `git diff` sebelum commit
- Pakai `git log --oneline --graph` untuk overview cepat

5.9 Cheat Commands

```
$ git log --oneline           # Riwayat ringkas
$ git log --oneline --graph  # Riwayat dengan grafik
$ git show <commit>         # Detail commit tertentu
$ git diff                   # Perubahan belum di-stage
$ git diff --staged          # Perubahan sudah di-stage
```


6. Repo Bersih: .gitignore

6.1 Kenapa Bab Ini Penting

Tidak semua file di folder proyek perlu dilacak Git. Ada file temporary, file konfigurasi lokal, file build, bahkan credential yang sangat tidak boleh di-commit. Kalau semua file ikut ter-commit, repository jadi berantakan dan bahkan bisa berbahaya (credential bocor!). `.gitignore` adalah cara Anda memberi tahu Git: "File-file ini jangan dilacak." Bab ini mengajarkan cara menjaga repository tetap bersih dan aman.

6.2 Tujuan Belajar

- Memahami file apa yang tidak boleh di-commit
- Membuat `.gitignore` dengan pola yang tepat
- Menggunakan pola wildcard untuk ignore banyak file
- Menghapus file terlanjur ter-track dengan `git rm --cached`
- Memahami keamanan: jangan commit credential

6.3 Konsep Kunci

File yang Tidak Boleh Masuk Repository

Ada beberapa kategori file yang sebaiknya di-ignore:

1. File Temporary/Cache

Contoh: `~$file.docx`, `*.tmp`, `*.log`, folder `__pycache__`/, `node_modules`/.

Kenapa? File ini dihasilkan otomatis dan tidak ada gunanya di-commit. Setiap orang bisa generate sendiri.

2. File Konfigurasi Lokal

Contoh: `.vscode/settings.json` (konfigurasi pribadi), `.env` (environment variables lokal).

Kenapa? Setiap developer punya preferensi berbeda. Konfigurasi lokal tidak perlu dibagikan.

3. File Build/Output

Contoh: `dist/`, `build/`, `.exe`, `.class`.

Kenapa? File hasil kompilasi bisa di-generate ulang dari source code. Tidak perlu disimpan di repository.

4. Credential dan Secret

Contoh: `credentials.json`, `secret.key`, `.env` yang berisi API key.

Kenapa? **Sangat berbahaya**. Kalau ter-commit dan repo public, orang lain bisa pakai credential Anda. Bahkan di repo private, ini tidak aman.

5. File Besar yang Tidak Relevan

Contoh: `video.mp4`, `dataset.csv` (gigabytes).

Kenapa? Git tidak dirancang untuk file besar. Repository jadi berat dan lambat. Untuk penanganan file besar, git menggunakan Git LFS (Large File Storage). Tetapi hal ini tidak dibahas di buku ini.

Aturan Singkat

Commit kode sumber & config (yang berbasis teks); jangan commit output build, secrets, atau dependencies.

Pola `.gitignore`

File `.gitignore` berisi daftar pola file yang akan di-ignore Git. Satu pola per baris.

Masuk Repo	Jangan Masuk
Kode sumber (.html, .js, .py, .java)	.env / credentials / API keys
Config proyek (package.json, pom.xml)	Build output (dist/, target/, build/)
Dokumentasi (.md, docs)	node_modules, vendor/
.gitignore	File temp OS (.DS_Store, thumbs.db)
Assets ringan (icons, svg kecil)	Database dumps, dataset besar
Sample data kecil	Cache, log files (*.log)

Tabel 6.1: *Tabel file yang boleh dan tidak boleh di-commit ke Git repository*

Pola umum:

```
# Komentar (diabaikan)
file.txt          # Ignore file bernama "file.txt"
*.log             # Ignore semua file dengan ekstensi .log
temp/             # Ignore folder "temp" dan isinya
!important.log    # Exception: jangan ignore file ini (meski pola
                  sebelumnya cocok)
```

Wildcard:

- *: Cocok dengan karakter apapun (kecuali /)
- **: Cocok dengan folder nested

Contoh:

```
logs/*.log        # Ignore file .log di folder logs (tapi tidak
                  di subfolder)
logs/**/*.log     # Ignore file .log di logs dan semua
                  subfoldernya
```

Kapan Membuat `.gitignore`?

Sejak awal proyek. Idealnya, file `.gitignore` dibuat di commit pertama atau kedua. Kalau file sudah terlanjur ter-commit, Anda harus untrack secara manual (akan kita praktikkan).

6.4 Praktik CLI: Membuat dan Menggunakan `.gitignore`

1. Membuat File yang Akan Di-Ignore

Pertama, kita buat beberapa file yang seharusnya di-ignore, jalankan perintah berikut di terminal Git Bash:

```
$ cd ~/web-tanpa-drama
$ mkdir notes
$ echo "Catatan pribadi" > notes/temp.txt
$ echo "Log file" > debug.log
$ echo "API_KEY=12345secret" > .env
```

File dan folder di atas mensimulasikan skenario nyata: catatan pribadi, file log, dan file credential.

- Folder `notes/` dengan file `temp.txt` (catatan pribadi, tidak perlu di-commit)
- File `debug.log` (temporary log)
- File `.env` (credential berupa password database atau secret key, sangat berbahaya kalau ter-commit!)

2. Cek Status dengan `git status`

```
$ git status
```

Output:

```
On branch main
Untracked files:
```

```
(use "git add <file>..." to include in what will be
    committed)
    .env
    debug.log
    notes/

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

Git melihat semua file ini sebagai untracked. Kalau kita tidak hati-hati dan ketik `git add .`, semuanya akan masuk staging , termasuk `.env` yang berisi credential!

3. Buat File `.gitignore`

Sekarang kita buat `.gitignore` untuk melindungi diri:

```
$ cat > .gitignore << 'EOF'
# Temporary files
*.log
*.tmp

# Environment variables (credentials)
.env

# Personal notes
notes/

# Build output (nanti kalau ada)
dist/
build/
EOF
```

4. Cek status kembali

```
$ git status
```

Output:

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

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

Luar biasa! File `debug.log`, `.env`, dan folder `notes/` sudah hilang dari output. Git meng-ignore mereka. Yang tersisa hanya `.gitignore` itu sendiri, file ini **harus di-commit** agar tim Anda juga dapat perlindungan yang sama.

5. Commit file `.gitignore`

```
$ git add .gitignore
$ git commit -m "chore: add gitignore for security and
  cleanliness"
```

Output:

```
[main e8f9a2b] chore: add gitignore for security and
  cleanliness
1 file changed, 11 insertions(+)
create mode 100644 .gitignore
```

Catatan: `chore:` adalah prefix untuk perubahan yang tidak menambah fitur atau fix bug, tapi penting untuk maintenance proyek.

6. Verifikasi apakah ignore bekerja

Coba buat file baru yang cocok dengan pola ignore:

```
$ echo "Another log" > app.log
$ git status
```

Output:

```
On branch main
nothing to commit, working tree clean
```

Git tidak melihat `app.log` karena pola `*.log` di `.gitignore`. Berhasil!

7. Kasus file terlanjur ter-track

Bayangkan Anda lupa bikin `.gitignore` dan sudah commit `debug.log`. Sekarang Anda buat `.gitignore`, tapi `debug.log` masih ter-track. Bagaimana solusinya?

Simulasi kasus ini:

```
$ echo "Old log file" > old.log
$ git add old.log
$ git commit -m "Wrong commit: add log file"
```

Sekarang `old.log` ter-track. Kita tambahkan `old.log` ke `.gitignore`:

```
$ echo "old.log" >> .gitignore
$ git status
```

Output:

```
On branch main
Changes not staged for commit:
    modified:   .gitignore

Untracked files:
```

```
(use "git add <file>..." to include in what will be
    committed)
    old.log
```

Tunggu, `old.log` masih muncul? Ya, karena file yang sudah ter-track tidak otomatis berhenti dilacak hanya karena ditambahkan ke `.gitignore`.

Solusinya: untrack file tersebut dengan `git rm --cached`:

```
$ git rm --cached old.log
```

Output:

```
$ rm 'old.log'
```

Command ini menghapus `old.log` dari staging/tracking Git, tapi **tidak menghapus file dari disk**. File masih ada di folder Anda.

Sekarang cek status:

```
$ git status
```

Output:

```
On branch main
Changes to be committed:
    modified:   .gitignore
    deleted:    old.log
```

File `old.log` sekarang "deleted" dari sisi Git (untracked), tapi masih ada di disk. Commit perubahan ini:

```
$ git commit -m "chore: stop tracking old.log"
```

Setelah ini, `old.log` benar-benar di-ignore.

6.5 Praktik VS Code: Melihat File Ter-Ignore

1. Buka Folder di VS Code

Buka `web-tanpa-drama` di VS Code.

2. Lihat explorer di VS Code dan amati!

Di Explorer sidebar, Anda akan melihat file seperti `debug.log`, `.env`, dan folder `notes/` dengan warna abu-abu redup (dimmed). Ini menandakan file tersebut di-ignore oleh Git.

Kalau Anda buka panel Source Control, file-file tersebut tidak muncul di "Changes".

3. Edit `.gitignore`

Buka `.gitignore` di VS Code. Anda bisa tambahkan pola baru kapan saja.

Misalnya, tambahkan:

```
# VS Code settings
.vscode/
```

Save file. Sekarang folder `.vscode/` (kalau ada) akan di-ignore.

6.6 Kesalahan Umum

1. **Lupa buat `.gitignore` sejak awal** File temporary atau credential sudah terlanjur ter-commit. Solusi: Buat `.gitignore` sekarang, lalu pakai `git rm --cached` untuk untrack file yang salah. Kalau credential sudah ter-push ke GitHub, **ganti credential tersebut segera**, dianggap sudah bocor.
2. **Commit `.env` atau credential lainnya** Ini kesalahan fatal. Kalau repository public, orang bisa pakai API key Anda, akses database, dll. Solusi: Jangan pernah commit credential.

Pakai `.gitignore` dan template `.env.example` (tanpa value sebenarnya) sebagai gantinya.

3. **Pakai pola ignore yang terlalu luas** Misalnya `*` (ignore semua). Repository jadi kosong. Solusi: Hati-hati dengan wildcard. Test dulu dengan `git status` setelah menambahkan pola baru.
4. **Tidak commit `.gitignore` itu sendiri** Tim Anda tidak dapat perlindungan yang sama. Solusi: `.gitignore` harus di-commit agar semua orang dapat pattern yang sama.
5. **Bingung kenapa file tidak ter-ignore padahal sudah di `.gitignore`** File tersebut sudah ter-track sebelumnya. Solusi: Pakai `git rm --cached <file>` untuk untrack.
6. **Pakai `git rm` tanpa `--cached`** File terhapus dari disk juga, bukan hanya dari Git. Solusi: Selalu pakai `git rm --cached` kalau Anda ingin untrack tapi tetap simpan file.

6.7 Latihan

Instruksi

1. Buat folder `build/` dan file `build/output.txt` di dalamnya.
2. Buat file `secret.key` dengan isi bebas (simulasi credential).
3. Tambahkan kedua item di atas ke `.gitignore`: “`build/ secret.key`”
4. Verifikasi dengan `git status`, kedua item tidak boleh muncul.
5. Commit perubahan `.gitignore` dengan pesan `chore: ignore build folder and secret key`.

Hasil yang Diharapkan

Output `git status` setelah langkah 4 seharusnya hanya menampilkan `.gitignore` sebagai file yang berubah. File `secret.key` dan folder `build/` tidak muncul karena sudah di-ignore.

Setelah commit, `git status` menunjukkan "nothing to commit, working tree clean".

6.8 Ringkasan

- **.gitignore** memberi tahu Git file/folder mana yang tidak perlu dilacak.
- **Pola umum:**
 - `*.log`: Ignore semua file `.log`
 - `folder/`: Ignore folder dan isinya
 - `!exception.log`: Jangan ignore file ini
- **File yang harus di-ignore:**
 - Temporary files (`.tmp`, `.log`)
 - Build output (`dist/`, `build/`)
 - Credential (`.env`, `secret.key`)
 - Konfigurasi lokal (`.vscode/`, IDE settings)
- **`git rm --cached <file>`**: Untrack file yang sudah terlanjur ter-track tanpa menghapus file dari disk.
- **Keamanan**: Jangan pernah commit credential. Kalau terlanjur, ganti credential segera.

`.gitignore` adalah pelindung pertama repository Anda. Biasanya cek `git status` sebelum `git add .` untuk memastikan tidak ada file yang tidak seharusnya masuk staging.

Di bab berikutnya, kita akan mulai petualangan branch, cara kerja di jalur terpisah tanpa merusak kode utama!

6.9 Cheat Commands

```
# Buat .gitignore
$ cat > .gitignore << 'EOF'
...pola ignore...
EOF

# Untrack file yang terlanjur ter-track
$ git rm --cached <file>

# Lihat apa yang akan di-ignore (before commit)
$ git status # File ter-ignore tidak muncul

# Commit .gitignore
$ git add .gitignore
$ git commit -m "chore: add gitignore"
```

Repository Anda sekarang bersih dan aman. Selamat!

LAMPIRAN



A. Cheat Sheet

A.1 Perintah Harian

Setup & Konfigurasi

```
git --version
git config --global user.name "Nama Anda"
git config --global user.email "email@example.com"
git config --list
```

Inisialisasi & Status

```
git init                # Buat repo baru
git status              # Cek status (paling sering!)
git status -s          # Status ringkas
```

Staging & Commit

```
git add <file>          # Stage file spesifik
git add .               # Stage semua (hati-hati!)
git restore --staged <file> # Unstage
git commit -m "message" # Commit dengan pesan
git commit -am "message" # Stage + commit (file tracked)
```

History & Perubahan

```
git log                # Log commit
git log --oneline      # Log ringkas
git log --oneline --graph # Log dengan graph
git log --oneline -5   # 5 commit terakhir
git diff              # Perubahan belum stage
git diff --staged     # Perubahan sudah stage
git diff <hash>       # Perubahan vs commit tertentu
```

Branch & Switch

```
git branch                # Lihat daftar branch
git branch -a             # Include remote branch
git switch -c <nama>      # Buat dan switch
git switch <nama>         # Pindah branch
git switch -              # Pindah ke branch sebelumnya
git branch -m <nama-baru> # Rename branch saat ini
git branch -d <nama>      # Hapus branch (merged)
git branch -D <nama>      # Hapus paksa (belum merged)
```

Merge

```
git merge <branch>      # Merge branch ke branch saat ini
git merge --abort        # Batalkan merge
git branch --merged      # Lihat branch merged
```

Remote (GitHub)

```
git remote add origin <URL> # Tambah remote
git remote -v                # Lihat remote
git remote set-url origin <URL> # Ganti URL
git fetch origin              # Download info tanpa merge
git pull origin main          # Fetch + merge
git pull                      # Jika tracking terset
git push -u origin main       # Push pertama (set upstream)
git push                      # Push biasa
git push origin --delete <branch> # Hapus branch remote
```

Stash

```
git stash push -m "WIP"      # Simpan perubahan
git stash list                # Lihat daftar
git stash pop                 # Apply + hapus
git stash apply stash@0      # Apply tanpa hapus
```

```
git stash drop stash@0    # Hapus satu
git stash clear           # Hapus semua (hati-hati!)
```

Undo & Restore

```
git restore <file>        # Buang perubahan belum stage
git restore --staged <file> # Unstage
git revert HEAD            # Revert commit (aman untuk remote)
git revert <hash>          # Revert commit spesifik
# Hapus commit, perubahan di staging
git reset --soft HEAD~1
# Hapus commit & staging (default)
git reset --mixed HEAD~1
# Hapus commit & perubahan (HATI-HATI!)
git reset --hard HEAD~1
# Lihat history HEAD (recover commit)
git reflog
```

Troubleshooting

```
git branch -vv            # Cek tracking
git status --ignored      # Lihat file ignored
# Hapus untracked files (hati-hati!)
git clean -fd
git merge --abort          # Batalkan merge
git switch main            # Kembali dari detached HEAD
```

A.2 GitHub CLI (gh)

Authentication & Setup

```
gh auth login             # Login ke GitHub
gh auth status            # Cek status auth
gh auth refresh           # Refresh token
```

Repository

```
gh repo create          # Buat repo (interaktif)
gh repo create <nama>   # Buat repo langsung
gh repo view            # Lihat info repo
gh repo list            # List repo Anda
gh repo clone <user/repo> # Clone repo
```

Pull Requests

```
gh pr create            # Buat PR (interaktif)
gh pr list              # List PR
gh pr view <number>     # Lihat detail PR
gh pr checkout <number> # Checkout PR secara lokal
gh pr merge <number>    # Merge PR
gh pr close <number>    # Tutup PR
```

Issues

```
gh issue create         # Buat issue
gh issue list           # List issues
gh issue view <number>  # Lihat issue
gh issue close <number> # Tutup issue
```

A.3 Workflow Cepat

Setup Repo Baru

```
mkdir project-name && cd project-name
git init
git config user.name "Nama"
git config user.email "email@example.com"
echo "# Project" > README.md
git add README.md
git commit -m "feat: initial commit"
gh repo create project-name --public
```

```
git remote add origin https://github.com/user/project-name.git
git push -u origin main
```

Clone & Siap Kerja

```
gh repo clone user/project
cd project
git switch -c feature/new-feature
# ... edit files ...
git add .
git commit -m "feat: add new feature"
git push -u origin feature/new-feature
gh pr create
```

Branching Workflow

```
git switch main
git pull
git switch -c feature/xyz
# ... kerja di branch feature/xyz ...
git add .
git commit -m "feat: implement xyz"
git push -u origin feature/xyz
gh pr create
# Setelah PR di-review dan merged:
git switch main
git pull
git branch -d feature/xyz
```

Fix Bug Cepat

```
git switch main
git pull
git switch -c fix/bug-name
# ... fix bug ...
```

```
git add .
git commit -m "fix: resolve bug-name"
git push -u origin fix/bug-name
gh pr create
```

A.4 Tips Produktivitas

Alias Git (Opsional)

Tambahkan ke ~/.gitconfig:

```
[alias]
    st = status -s
    co = checkout
    br = branch
    lg = log --oneline --graph --all
    last = log -1 HEAD
    unstage = restore --staged
    undo = reset --soft HEAD~1
```

Lalu pakai:

```
git st      # singkat status
git lg      # log cantik
git unstage . # unstage semua
```

B. Glosarium

Daftar istilah Git dengan definisi singkat dan contoh. Format:
Istilah - Definisi - Contoh.

A-C

- **Add:** Memasukkan perubahan file ke staging area. Contoh:
`git add index.html`
 - **Branch:** Jalur kerja terpisah untuk develop fitur tanpa ganggu kode stabil. Branch adalah pointer ke commit. Contoh: `git switch -c feature/xyz`
 - **Clone:** Salin seluruh repo remote (dengan history) ke lokal. Contoh: `git clone https://github.com/user/repo.git`
 - **Commit:** Snapshot perubahan dengan pesan dan hash unik. Seperti “save game”. Contoh: `git commit -m "feat: add homepage"`
 - **Conflict:** Git tidak bisa auto-merge karena dua branch ubah baris yang sama. Harus resolve manual. Marker: <<<<<<<<, =====, >>>>>>>>
-

D-H

- **Detached HEAD:** HEAD menunjuk ke commit tertentu, bukan branch. Commit baru tidak tercatat di branch manapun. Fix: `git switch main`
- **Diff:** Lihat perbedaan antara dua versi file/commit. Contoh: `git diff`, `git diff:-staged`
- **Fast-forward:** Merge dengan hanya geser pointer branch

(tidak buat merge commit). Terjadi kalau tidak ada divergent commit.

- **Fetch:** Ambil info terbaru dari remote tanpa ubah working directory. Seperti “cek update” tanpa install. Contoh: `git fetch origin`
 - **.gitignore:** File berisi pola file/folder yang diabaikan Git. Contoh: `node_modules/, .env, *.log`
 - **HEAD:** Pointer ke posisi Anda saat ini (biasanya ke branch). `HEAD~1` = commit sebelum HEAD.
 - **History:** Timeline semua commit. Bisa lihat dengan `git log`.
-

I-P

- **Init:** Buat repository Git baru di folder lokal. Contoh: `git init`
- **Log:** Lihat history commit. Contoh: `git log:-oneline:-graph`
- **Main (atau Master):** Branch utama. Nama default GitHub sekarang main, repo lama pakai master.
- **Merge:** Gabungkan dua branch. Git buat merge commit dengan 2 parent. Contoh: `git merge feature/xyz`
- **Merge Commit:** Commit hasil merge dengan 2 parent. Terlihat di `git log:-graph` sebagai percabangan.
- **Origin:** Nama default untuk remote utama (biasanya URL GitHub). Contoh: `git remote add origin <URL>`
- **Pull:** Fetch + merge. Ambil update remote dan langsung merge ke branch lokal. Contoh: `git pull origin main`
- **Pull Request (PR):** Fitur platform (GitHub) untuk minta review code sebelum merge. Bukan fitur Git, tapi workflow

standar.

- **Push:** Kirim commit lokal ke remote. Contoh: `git push origin main`
-

R-S

- **Rebase** - Pindahkan commit ke ujung branch target. History jadi linear tapi commit hash berubah (rewrite history). Contoh: `git rebase main`
- **Reflog** - Log internal Git untuk semua pergerakan HEAD. Berguna untuk menemukan commit “hilang”. Contoh: `git reflog`
- **Remote** - Versi repo di server (cloud) sebagai backup dan pusat kolaborasi. Contoh: GitHub, GitLab.
- **Repository (Repo)** - Folder project dengan semua file, history commit, dan metadata di folder `.git`.
- **Reset** - Mundur ke commit tertentu, hapus commit setelahnya. Ada 3 mode: `--soft`, `--mixed`, `--hard`. Contoh: `git reset --soft HEAD~1`
- **Restore** - Batalkan perubahan file di working directory atau staging. Contoh: `git restore <file>`, `git restore --staged <file>`
- **Revert** - Buat commit baru yang batalkan commit lama. Commit lama tetap ada—aman untuk remote. Contoh: `git revert HEAD`
- **Staging Area (Index)** - Area sementara tempat file menunggu sebelum commit. File harus di-stage (`git add`) dulu.
- **Stash** - Penyimpanan sementara untuk perubahan belum commit. Berguna saat harus pindah branch. Contoh: `git stash push -m "WIP"`

- **Status** - Lihat kondisi working directory dan staging area.
Contoh: `git status`
 - **Switch** - Pindah antar branch. Contoh: `git switch main`,
`git switch -c feature/xyz`
-

T-W

- **Tag** - Label tetap untuk commit (untuk versioning). Contoh:
`git tag v1.0.0`
- **Tracking Branch** - Branch lokal yang terhubung dengan branch remote. Setelah set tracking, cukup `git push` tanpa tulis origin main. Contoh: `git push -u origin main`
- **Untracked Files** - File di working directory yang belum pernah di-`git add`. Git “lihat” tapi tidak track perubahan.
- **Upstream** - Branch remote yang di-track oleh branch lokal. Contoh: branch lokal main track upstream `origin/main`.
- **Working Directory (Working Tree)** - Folder project tempat Anda edit file—kondisi “saat ini” sebelum stage/commit. “Working tree clean” = tidak ada perubahan belum commit.

Daftar Pustaka

Buku ini disusun berdasarkan dokumentasi resmi, tutorial, dan best practices dari berbagai sumber terpercaya berikut:

Dokumentasi Resmi

1. Git Documentation

Git Official Documentation

<https://git-scm.com/doc>

Dokumentasi resmi Git yang mencakup command reference, user manual, dan ProGit book.

2. GitHub Docs

GitHub Official Documentation

<https://docs.github.com>

Dokumentasi lengkap penggunaan GitHub, Pull Request workflow, dan kolaborasi tim.

3. GitHub CLI Manual

GitHub CLI (gh) Documentation

<https://cli.github.com/manual/>

Dokumentasi GitHub CLI untuk operasi GitHub via command line.

Tutorial dan Dokumentasi Ofisial

4. Atlassian Git Tutorials

Atlassian Bitbucket Git Tutorials

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

Tutorial komprehensif Git dari Atlassian, mencakup:

- Using Branches: <https://www.atlassian.com/git/tutorials/using-branches>
- Git Merge: <https://www.atlassian.com/git/tutorials/using-branches/git-merge>
- Git Rebase: <https://www.atlassian.com/git/tutorials/rewriting-history/git-rebase>
- Resolving Conflicts: <https://www.atlassian.com/git/tutorials/using-branches/merge-conflicts>

5. Git Handbook - GitHub

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

Dokumentasi singkat dan praktis untuk memulai Git.

6. Learn Git Branching

<https://learngitbranching.js.org/>

Pembelajaran interaktif visual untuk memahami Git branching dan merging.

Best Practices dan Konvensi

7. Conventional Commits

<https://www.conventionalcommits.org/>

Spesifikasi konvensi pesan commit (feat:, fix:, docs:, dll.) yang digunakan dalam buku ini.

8. Git Best Practices - GitLab

<https://about.gitlab.com/topics/version-control/version-control-best-practices/>

Best practices version control dan Git workflow.

Buku Referensi

9. Pro Git Book (2nd Edition)

Scott Chacon & Ben Straub

Apress, 2014

<https://git-scm.com/book/en/v2>

Buku komprehensif Git (tersedia gratis online), digunakan sebagai referensi konsep dasar Git.

10. Git Pocket Guide

Richard E. Silverman

O'Reilly Media, 2013

Pedoman ringkas Git command dan workflow.

Tools dan Integrasi

11. Visual Studio Code - Using Git

<https://code.visualstudio.com/docs/sourcecontrol/overview>

Dokumentasi integrasi Git dengan VS Code.

12. Git for Windows

<https://gitforwindows.org/>

Git implementation untuk Windows, termasuk Git Bash.

Visualisasi dan Diagram

Diagram dan ilustrasi workflow Git dalam buku ini terinspirasi dari:

- Atlassian Git Tutorials visualization
- Pro Git Book diagrams
- Git-scm.com documentation graphics

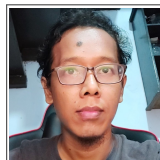
Tentang Penulis

Dian Hanifudin Subhi adalah akademisi sekaligus praktisi di bidang Teknologi Informasi, serta menyelesaikan pendidikan S1 dan S2 di Institut Teknologi Sepuluh Nopember (ITS). Ketertarikannya berada di bidang pemrograman, Quality Assurance (QA), dan cloud computing mulai dari pengembangan aplikasi, penerapan praktik pengujian yang rapi dan terukur, hingga membangun layanan yang siap berjalan di lingkungan cloud. Penulis dapat dihubungi melalui: dhanifudin.com.



Usman Nurhasan adalah seorang akademisi dan peneliti di bidang Teknologi Informasi yang berafiliasi dengan Politeknik Negeri Malang. Ia secara aktif mengembangkan riset terapan pada sistem cerdas, Internet of Things (IoT), augmented reality, serta aplikasi berbasis mobile dan web. Publikasi yang dihasilkan telah berkontribusi pada inovasi teknologi pendidikan, sistem monitoring, dan pengembangan aplikasi cerdas yang berorientasi pada solusi praktis bagi masyarakat. Saat ini, Penulis juga memfokuskan penelitian pada pengembangan model penilaian kualitas kode Python. Riset ini bertujuan untuk merumuskan kerangka evaluasi yang sistematis dan terukur, sehingga kualitas perangkat lunak dapat dinilai secara objektif berdasarkan aspek sintaksis, semantik, efisiensi, dan keterbacaan.

Putra Prima Arhandi lahir di Pekanbaru dan menyelesaikan S1 di Fakultas Teknik Universitas Brawijaya, S2 di Teknik Informatika Institut Teknologi Sepuluh Nopember Surabaya. Penulis merupakan pengajar mata kuliah Pemrograman Mobile, Pemrograman Web dan Pengujian Perangkat Lunak di Jurusan Teknologi Informasi, Politeknik Negeri Malang, sejak tahun 2015 sampai sekarang. Aktif membuat konten pembelajaran di youtube melalui channel DosenNgoding dan aktif kontribusi ke open source project di github @siubie, Semangat Ngoding !!!



Sofyan Noor Arief merupakan seorang dosen junior di JTI POLINEMA yang berfokus pada bidang keilmuan Jaringan Komputer dan juga seorang freelancer jasa konsultan dan pemasangan jaringan. Lulusan D3 Manajemen Informatika POLINEMA, D4-LJ Teknik Informatika PENS, dan S2 Teknik Informatika ITS. Punya hobi otak-atik server dan perangkat-perangkat jaringan lainnya. Menyukai dan menekuni riset pada bidang distributed computing yang diimplementasikan pada perangkat-perangkat low-power computing. Penulis dapat dihubungi melalui: sofyan@polinema.ac.id.

Sinopsis

Git Tanpa Drama membawa Anda dari pemula absolut hingga praktisi Git yang percaya diri. Buku ini menghilangkan kompleksitas Git melalui pendekatan praktis dengan bahasa yang mudah dipahami.

Melalui 14 bab terstruktur, Anda akan menguasai: fondasi Git (`init`, `commit`, `log`, `.gitignore`), branching & merging, kolaborasi GitHub dengan Pull Request, hingga advanced operations (`stash`, `rebase`, dan `undo strategies`). Setiap konsep dijelaskan dengan satu proyek praktis dari folder kosong hingga kolaborasi tim profesional.

Cocok untuk: Mahasiswa IT/CS, programmer pemula, developer self-taught, dan tim yang ingin mengadopsi workflow Git profesional.

Saatnya bekerja tanpa drama, dengan Git sebagai senjata andalan Anda.

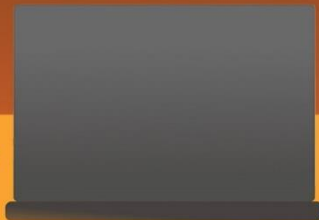
Git Tanpa Drama adalah panduan praktis untuk menguasai Git dan GitHub dari nol hingga kolaborasi profesional, tanpa kebingungan, tanpa panik, tanpa drama.

Dalam 14 bab, Anda telah belajar:

- **Fondasi kokoh:** Dari git init hingga commit yang rapi, membaca history dengan log dan diff, mengelola file dengan .gitignore
- **Branching & Merging:** Bekerja dengan branch, merge tanpa takut, menyelesaikan konflik dengan percaya diri
- **Kolaborasi GitHub:** *Push ke remote*, menangani *push* ditolak, *Pull Request workflow* yang profesional
- **Advanced & Recovery:** Stash untuk pindah konteks, memahami *rebase vs merge*, *undo* dengan *restore/revert/reset*

Satu proyek praktis dari awal hingga akhir. Dari folder kosong hingga kolaborasi tim di GitHub. Git bukan musuh. Git adalah alat yang akan menemani Anda sepanjang karier. Dengan fondasi yang Anda bangun dari buku ini, Anda siap bekerja menghadapi project nyata, tim profesional, dan tantangan kolaborasi modern.

Selamat! Anda bukan lagi pemula Git.
Tetap praktik, tetap tenang, tetap tanpa drama.



git

KBM
INDONESIA
www.penerbitkbm.com

PENERBIT KBM INDONESIA
Anggota IKAPI
0813 5751 7526 / (0353) 3234874
Kantor : Maguwoharjo, Depok, Sleman-Yogyakarta
@penerbitbukujogja
@penerbit.kbm

ISBN 978-634-278-173-9



9

786342

781739