Version Control (Git)

Maulana Hafidz Ismail

09 Oktober 2025

Contents

1 Software Collaboration

1.1 Pentingnya Komunikasi

- Komunikasi adalah keterampilan utama untuk berkolaborasi dengan pengembang lain dan memastikan pemahaman yang konsisten tentang persyaratan produk.
- Kolaborasi yang efektif diperlukan untuk bekerja pada proyek besar dengan berbagai keterampilan.

1.2 Kolaborasi dengan Tim

- Pengembang harus bekerja sama dengan manajer produk, peneliti pengguna, dan desainer untuk membangun fitur yang tepat.
- Memprioritaskan pekerjaan dan memberikan konteks yang tepat kepada rekan kerja sangat penting untuk kemajuan proyek.

1.3 Pengembangan Keterampilan

- Keterampilan yang dibutuhkan dapat bervariasi antara perusahaan besar dan startup; di perusahaan besar, pengembang lebih terfokus, sedangkan di startup, mereka mungkin harus mengenakan banyak topi.
- Belajar untuk beradaptasi dengan preferensi kerja orang lain, seperti cara belajar visual atau berbicara, adalah bagian dari kolaborasi yang sukses.

1.4 System Version Control

- `Version Control` adalah sistem yang mencatat semua perubahan dan modifikasi pada file untuk tujuan pelacakan.
- Tujuan utama dari sistem kontrol versi adalah untuk menjaga jejak perubahan, memungkinkan pengembang untuk mengakses seluruh riwayat perubahan dan mengembalikan ke keadaan sebelumnya jika diperlukan.

1.5 Manfaat Version Control

- Riwayat revisi memberikan catatan semua perubahan dalam proyek, memungkinkan pengembang untuk kembali ke titik stabil jika ada masalah.
- Identitas pengguna yang melakukan perubahan dicatat, sehingga tim dapat melihat siapa yang membuat perubahan dan kapan.
- Pelacakan Perubahan: `Version control systems` membantu pengembang melacak perubahan pada kode dan memastikan bahwa semua anggota tim memiliki akses ke versi terbaru.

- Sumber Kebenaran: Memiliki satu sumber kebenaran untuk semua perubahan historis sangat penting saat bekerja dalam tim, sehingga memudahkan kolaborasi.

1.6 Kolaborasi dan Efisiensi Version Control

- `Version Control System` mendukung kolaborasi tim dengan memungkinkan pengembang untuk mengajukan kode dan melacak perubahan yang perlu dilakukan.
- Proses tinjauan sejawat (`peer review`) membantu dalam mendapatkan umpan balik dari anggota tim lain sebelum kode diterapkan.

1.7 Peran dalam DevOps

- `Version Control` berperan penting dalam praktik `DevOps`, yang meningkatkan kemampuan organisasi untuk mengirimkan aplikasi atau layanan dengan kualitas tinggi dan kecepatan tinggi.
- Dengan kontrol versi, tim dapat bekerja lebih efisien dan memastikan bahwa fitur baru tidak merusak alur kerja yang sudah ada.

1.8 Kolaborasi di Meta

- Di Meta, para `engineer` memimpin setiap proyek dan berkoordinasi dengan `product data scientists` dan `researchers` mengenai apa yang sedang dibangun dan `timeline`-nya.
- Komunikasi dilakukan melalui pesan, dokumen, dan pertemuan untuk menyelesaikan masalah dan berbagi informasi.

1.9 Version Control di Meta

- Meta menggunakan satu `monolithic repository` untuk semua kode, yang memungkinkan berbagi dan penggunaan kembali kode antar tim.
- Setiap `engineer` di Meta dapat melakukan perubahan pada kode, menciptakan budaya tanggung jawab bersama.

1.10 Tantangan dalam Kolaborasi

- `Merge conflicts` sering terjadi karena banyaknya `engineer` yang bekerja dalam `repository` yang sama, sehingga tim berusaha untuk menulis perubahan yang lebih kecil dan menambahkan pengujian untuk mencegah masalah di produksi.
- Alat seperti "`get blame`" digunakan untuk melihat riwayat revisi file, membantu `engineer` memahami kode yang ditulis oleh orang lain dan menghubungi mereka jika ada pertanyaan.

1.11 Centralized Version Control Systems (CVCS)

- `CVCS` memiliki `server` dan `klien`, di mana `server` menyimpan `repositori` utama yang berisi seluruh riwayat versi dari `code base`. Perlu koneksi online.
- Pengembang perlu menarik (`pull`) kode dari `server` ke mesin lokal mereka untuk bekerja, dan semua perubahan harus didorong (`push`) kembali ke `server` agar dapat dilihat oleh pengembang lain.

1.12 Distributed Version Control Systems (DVCS)

- `DVCS` memungkinkan setiap pengguna memiliki salinan lengkap dari riwayat perubahan di sistem lokal mereka, sehingga dapat bekerja secara `offline`.
- Pengguna hanya perlu terhubung ke `server` untuk menarik (`pull`) perubahan terbaru atau mendorong (`push`) perubahan mereka sendiri.

1.13 Keuntungan dan Kerugian Version Control System

- Keuntungan `CVCS`: Lebih mudah dipelajari dan memberikan kontrol akses yang lebih baik kepada pengguna.
- Kerugian `CVCS`: Memerlukan koneksi ke `server` untuk melakukan hampir semua tindakan, yang dapat memperlambat proses.
- Keuntungan `DVCS`: Pengguna dapat bekerja tanpa koneksi ke `server`, meningkatkan kecepatan dan kinerja.
- Kerugian `DVCS`: Mungkin lebih kompleks untuk dipelajari dibandingkan `CVCS`.

1.14 Perbedaan Utama Antara CVCS dan DVCS

1.14.1 Kepemilikan Riwayat

- `CVCS`: Hanya `server` yang memiliki riwayat lengkap dari semua versi. `Developer` hanya memiliki salinan terbaru dari kode. Untuk melihat riwayat, mereka harus terhubung ke `server`.
- `DVCS`: Setiap `developer` memiliki salinan lengkap dari seluruh riwayat proyek di mesin lokal mereka. Ini memungkinkan mereka untuk melihat riwayat, melakukan `revert`, atau bahkan membuat cabang (`branch`) tanpa harus terhubung ke `server`.

1.14.2 Koneksi ke Server

- `CVCS`: Setiap kali `developer` ingin melakukan tindakan seperti `commit` atau melihat riwayat, mereka harus terhubung ke `server`. Ini bisa menjadi kendala jika koneksi internet tidak stabil.
- `DVCS`: `Developer` dapat melakukan banyak tindakan secara `offline`, seperti `commit`, `revert`, dan membuat cabang. Mereka hanya perlu terhubung ke `server` untuk menarik (`pull`) atau mendorong (`push`) perubahan.

1.14.3 Pengelolaan Konflik

- `CVCS`: Konflik biasanya dihadapi saat melakukan `commit`, dan `developer` harus menyelesaikannya dengan mengakses versi di `server`.
- `DVCS`: `Developer` dapat menyelesaikan konflik di lokal mereka sebelum mendorong perubahan ke `server`, memberikan lebih banyak kontrol atas proses tersebut.

1.15 Sejarah Version Control

- Version Control dimulai pada tahun 1980-an, sebelum adanya Internet.
- Salah satu `VCS` pertama yang signifikan adalah `Concurrent Versions System (CVS)`, yang dikembangkan oleh Walter F. Tichy pada tahun 1986 dan dirilis secara publik pada tahun 1990.
- `CVS` menyimpan informasi tentang setiap file dalam struktur folder. Namun, `CVS` memiliki kekurangan, seperti tidak adanya `integrity checks` yang dapat menyebabkan data menjadi korup.

1.16 Pengembangan VCS Selanjutnya

- `Subversion (SVN)` dikembangkan oleh CollabNet pada tahun 2000 sebagai penerus `CVS`, dengan perbaikan dalam hal `integrity checks` dan dukungan untuk file biner.
- Meskipun `SVN` populer, ia menggunakan model `centralized VCS`, yang berarti semua operasi harus dilakukan melalui `server` pusat, yang dapat menghambat pengembangan jika `server` mengalami masalah.

1.17 Munculnya Distributed Version Control Systems

- Pada tahun 2005, dua proyek baru dimulai untuk mengembangkan `distributed version control systems`: `Mercurial` dan `Git`. Keduanya muncul sebagai respons terhadap masalah yang dihadapi dalam pengembangan kernel Linux.

- `Mercurial`, yang dikembangkan oleh Olivia Mackall, adalah `VCS` terdistribusi yang berkinerja tinggi dan mudah digunakan.
- `Git`, yang dikembangkan oleh Linus Torvalds, dirilis secara publik pada tahun 2007 dan menjadi sangat populer di komunitas `open-source`, terutama karena desain terdistribusinya dan dukungan dari platform seperti `GitHub`.

1.18 Workflow

- Menggunakan `Version Control` tanpa `workflow` yang tepat dapat menyebabkan kekacauan, seperti konflik saat dua pengembang mengedit file yang sama secara bersamaan.
- `Workflow` yang baik mencakup proses untuk menyelesaikan konflik dan sistem `peer review` untuk memastikan kode yang diubah diperiksa sebelum digabungkan.

1.19 Continuous Integration (CI)

- `CI (Continuous Integration)` digunakan untuk mengotomatiskan integrasi perubahan kode dari beberapa pengembang ke dalam satu aliran utama.
- Dengan menggabungkan perubahan kecil secara sering, `CI` mengurangi jumlah konflik penggabungan dan memastikan `build` tetap stabil dengan menjalankan tes otomatis.

1.20 Continuous Delivery (CD)

- `CD (Continuous Delivery)` adalah praktik lanjutan yang dibangun di atas `CI`, di mana `pipeline CD` mengotomatiskan proses persiapan aplikasi untuk `deployment`.
- Tujuan utama `CD` adalah memastikan aplikasi selalu dalam keadaan dapat `dideploy`, meskipun setelah banyak perubahan, dengan langkah manual untuk persetujuan sebelum rilis ke lingkungan produksi.

1.21 Continuous Deployment

- `Continuous Deployment` mengotomatiskan proses `deployment` aplikasi ke produksi tanpa intervensi manual.
- Setiap perubahan yang lulus tes otomatis akan langsung `dideploy`, memastikan pembaruan dan perbaikan segera tersedia untuk pelanggan.

1.22 Proses Kerja Tim Pengembang

- Contoh Kasus: Dalam pengembangan fitur baru untuk aplikasi `e-commerce`, pengembang membuat perubahan dan mengirimkan `pull request` untuk ditinjau oleh rekan-rekan mereka.
- Resolusi Konflik: Ketika beberapa pengembang membuat perubahan pada bagian kode yang sama, riwayat revisi membantu dalam mengidentifikasi dan menyelesaikan konflik.

1.23 Manfaat Riwayat Revisi

- Aksesibilitas: Riwayat revisi memungkinkan anggota tim untuk melihat siapa yang membuat perubahan, kapan, dan alasan di balik perubahan tersebut.
- Manajemen Perubahan: Riwayat ini juga membantu dalam mengelola dan menggabungkan perubahan untuk mencapai tujuan bisnis dengan tepat waktu.

1.24 Lingkungan Staging

- Lingkungan `staging` harus meniru lingkungan `production` untuk menguji kode dengan akurat sebelum dirilis.
- `Staging` digunakan untuk menguji fitur baru dan memungkinkan tim `QA` atau pemangku kepentingan untuk melihat dan menggunakan fitur tersebut sebelum peluncuran.

1.25 Pengujian di Staging

- Lingkungan `staging` adalah tempat yang baik untuk menjalankan berbagai jenis pengujian, termasuk `Unit testing`, `Integration testing`, dan `performance testing`.
- Meskipun `performance testing` dapat dilakukan di `production`, biasanya dilakukan di luar jam sibuk untuk menghindari dampak pada pengalaman pengguna.

1.26 Migrasi dan Perubahan Konfigurasi di Staging

- `Staging` adalah tempat yang ideal untuk menguji dan memverifikasi data `migrations`, dengan `snapshot` dari `production` untuk memastikan skrip migrasi tidak menyebabkan masalah.
- Perubahan konfigurasi juga dapat diuji di `staging` untuk mengidentifikasi potensi masalah sebelum diterapkan di `production`.

1.27 Lingkungan Produksi (Production)

- `Produksi` adalah lingkungan `live` yang harus bebas dari masalah yang seharusnya sudah terdeteksi dan diperbaiki di `staging`.
- `Downtime` di `production` dapat berdampak pada pendapatan, terutama untuk layanan yang berorientasi pelanggan, seperti `e-commerce`.

1.28 Keamanan dan Reputasi di Produksi

- Pembaruan perangkat lunak harus diperiksa untuk kerentanan keamanan sebelum diterapkan di `production`.
- Masalah di `production` dapat merusak reputasi perusahaan dan mengurangi kepercayaan pengguna.

2 Command Line

2.1 Interaksi dengan Komputer

- Interaksi dengan komputer berarti bertukar informasi, di mana komputer mengirim data kepada pengguna dan pengguna juga mengirim data ke komputer.
- Perangkat `input` termasuk `keyboard`, `mouse`, `microphone`, dan `kamera`, sedangkan perangkat `output` meliputi `speakers`, `monitors`, dan `headsets`.

2.2 Graphical User Interfaces (GUIs) dan Command Line

- `GUIs (Graphical User Interfaces)` memudahkan interaksi dengan perangkat, tetapi dapat membatasi kemampuan interaksi manusia-komputer.
- Interaksi dengan perangkat seperti ponsel dan komputer dilakukan melalui `Graphic User Interface (GUI)`, yang merupakan lapisan di atas perintah dasar.
- `Command line` adalah alternatif yang kuat, memungkinkan pengguna untuk melakukan tugas lebih cepat dan dengan potensi kesalahan yang lebih sedikit.

2.3 Perintah Dasar di Command Line

- `cd`: Mengubah direktori ke folder tertentu. Contoh: `cd /desktop` untuk menuju ke desktop.
- `touch`: Membuat file baru. Contoh: `touch example.txt` untuk membuat file kosong.
- `mkdir`: Membuat folder baru. Contoh: `mkdir myjsproject` untuk membuat folder baru.
- `history`: Menampilkan riwayat perintah yang telah diketik.

2.4 Sejarah Unix dan Linux

- `Unix` dikembangkan oleh Ken Thompson dan Dennis Ritchie di AT&T Labs pada tahun 1969, dan menjadi dasar bagi banyak sistem operasi modern.
- `Linux`, yang dikembangkan oleh Linus Torvalds, muncul kemudian dan banyak digunakan di platform `cloud`.

2.5 Perintah Dasar Unix

- `cd`: Mengubah direktori dalam sistem file.
- `ls`: Menampilkan isi direktori saat ini, dengan berbagai `flag` seperti `-l` untuk detail dan `-a` untuk menampilkan file tersembunyi.
- `pwd`: Menampilkan jalur lengkap dari direktori kerja saat ini.
- `cp`: Menyalin file atau folder dari satu lokasi ke lokasi lain.
- `mv`: Memindahkan file dari satu direktori ke direktori lain.

2.6 Perintah Bash yang Umum

- `cd`: Mengubah direktori.
- `ls`: Menampilkan isi direktori.
- `rm`: Menghapus file atau direktori.
- `mv`: Memindahkan file atau folder.
- `touch`: Membuat file kosong baru atau memperbarui `timestamp` file.
- `cp`: Membuat salinan file atau folder.
- `mkdir`: Membuat direktori baru.
- `pwd`: Menampilkan lokasi saat ini di `shell`.
- `cat`: Membaca atau menggabungkan isi file.
- `less`: Menampilkan isi file satu halaman sekaligus.
- `grep`: Mencari isi file atau folder.

2.7 Flags dan Man Pages

- Setiap perintah `Bash` memiliki `flags` untuk mengubah `output` perintah tersebut. Misalnya, menambahkan `flag -l` pada perintah `ls` untuk menampilkan daftar dengan format yang berbeda.
- `Man pages` adalah manual untuk setiap perintah yang mencakup semua `flags` dan opsi yang tersedia. Contoh: ketik `man ls` untuk melihat `man page` dari perintah `ls`.

2.8 Mengedit File dengan VI atau VIM

- `VI` adalah editor visual yang digunakan untuk mengedit dan menyimpan file. `VIM` adalah versi yang lebih baik dari `VI` dengan beberapa perbaikan.
- `VIM` memiliki beberapa `mode`:
  - `Normal mode`: Mode `default` untuk navigasi dan operasi lainnya.
  - `Insert mode`: Untuk mengedit isi file secara langsung.
  - `Command line mode`: Dapat diakses dengan mengetikkan tanda titik dua `:` di `Normal mode`.

2.9 Navigasi dan File Konfigurasi

- Menggunakan perintah `cd` untuk berpindah ke direktori `home` dan `ls -la` untuk menampilkan semua file, termasuk file tersembunyi.
- Fokus pada file `bashRC`, yang berisi konfigurasi yang dieksekusi saat terminal dibuka, seperti pengaturan warna dan riwayat perintah.
- File `bash profile` digunakan untuk variabel lingkungan (`env`), seperti pengaturan direktori Java atau Python.

2.10 Membuat Shell Script

- Menggunakan editor `Vim` untuk membuat file baru bernama `testshell.sh`.
- Menambahkan `shebang` (`#!/bin/bash`) di bagian atas file untuk menunjukkan bahwa ini adalah `bash script`.
- Menggunakan perintah `echo` untuk mencetak "Hello World" ke layar.

2.11 Menjalankan dan Mengatur Izin File

- Setelah membuat file, file tersebut tidak dapat dieksekusi. Menggunakan perintah `chmod 755` untuk mengubah izin file agar dapat dieksekusi.
- Menjalankan `script` dengan perintah `./testshell.sh`, yang akan mencetak "Hello World" di layar.

2.12 Simbol Izin File

- `r` (`read`): Simbol ini menunjukkan bahwa file dapat dibaca. Jika Anda memiliki izin `r`, Anda dapat membuka dan melihat isi file tersebut.
- `w` (`write`): Simbol ini menunjukkan bahwa file dapat ditulis. Dengan izin `w`, Anda dapat mengedit atau menghapus file.

- `x` (`execute`): Simbol ini menunjukkan bahwa file dapat dieksekusi. Jika Anda memiliki izin `x`, Anda dapat menjalankan file tersebut sebagai program atau `script`.

2.13 Representasi Angka untuk Izin File

- 4 mewakili izin `read` (`r`).
- 2 mewakili izin `write` (`w`).
- 1 mewakili izin `execute` (`x`).
- Kombinasi angka-angka ini digunakan untuk memberikan izin yang diinginkan. Misalnya:
  - 7 (4 + 2 + 1) berarti izin `read`, `write`, dan `execute`.
  - 6 (4 + 2) berarti izin `read` dan `write`.
  - 5 (4 + 1) berarti izin `read` dan `execute`.
  - 0 berarti tidak ada izin sama sekali.

2.14 Contoh Pengaturan Izin

- Ketika Anda menggunakan perintah `chmod 755 testshell.sh`, Anda mengatur izin file `testshell.sh` sebagai berikut:
  - 7 (untuk pemilik file): Ini berarti pemilik memiliki izin `r`, `w`, dan `x` (4 + 2 + 1 = 7).
  - 5 (untuk grup): Ini berarti grup memiliki izin `r` dan `x` (4 + 1 = 5), tetapi tidak dapat menulis.
  - 5 (untuk orang lain): Ini juga berarti orang lain memiliki izin `r` dan `x` (4 + 1 = 5), tetapi tidak dapat menulis.

2.15 Perintah pwd (print working directory)

- Perintah ini digunakan untuk memeriksa direktori tempat Anda berada saat ini.
- Misalnya, jika `output`-nya adalah `/`, itu artinya Anda berada di `root directory`, yaitu direktori tingkat teratas dalam sistem operasi.

2.16 Perintah ls (list)

- Perintah ini digunakan untuk melihat isi dari direktori yang sedang Anda buka.
- Jika Anda mengetik `ls` di `root directory`, Anda akan melihat daftar nama direktori yang ada di dalamnya.

- Anda bisa menambahkan `flag -l` ke perintah `ls` untuk menampilkan detail lebih lanjut dalam format daftar (`list`). Contohnya adalah `ls -l`.
- Dalam format daftar ini, Anda bisa mengidentifikasi jenis-jenis file atau direktori berdasarkan simbol di awal baris:
  - `l`: Mewakili `link file`. Contohnya `temp` yang merupakan `link` ke direktori `tmp`.
  - `d`: Mewakili `standard directory`. Contohnya, `bin`.
  - `-`: Mewakili `standard file` seperti file teks atau konfigurasi. Contohnya, `resolve.conf`.
- Anda juga akan melihat informasi tentang pemilik (`owner`) dan grup (`group`) yang terkait dengan setiap file atau direktori.

2.17 Perintah cd (change directory)

- Perintah ini digunakan untuk berpindah dari satu direktori ke direktori lain.
- Untuk masuk ke direktori tertentu, ketik `cd` diikuti dengan nama direktori tersebut, misalnya `cd etc`.
- Untuk kembali ke direktori induk (`parent directory`), gunakan `cd ..`. Contohnya, dari `etc` ke `root directory`.
- Anda juga bisa menggunakan `absolute path` untuk kembali ke direktori utama, contohnya `cd /`.
- Untuk melangkah maju melalui beberapa direktori secara berurutan, Anda bisa menggunakan perintah `cd` beberapa kali, seperti `cd etc` lalu `cd ssh`.

2.18 Perintah mkdir (Make Directory)

- Perintah ini membuat direktori/`folder` baru. Contoh: `mkdir submissions`.
- Dua direktori bisa dibuat sekaligus dengan dipisah spasi seperti `mkdir dir1 dir2`.
- Kita bisa langsung menentukan lokasi direktori barunya seperti ini `mkdir /existingDir/newDir`.
- Perintah `mkdir -p dir2/dir3` menggunakan `-p` untuk memastikan semua `folder` induk yang belum ada ikut dibuat.

2.19 Perintah mv (Move)

- Perintah ini memindahkan direktori ke direktori lain.
- Contoh: `mv submissions archive` memindahkan `submissions` ke dalam `archive`.
- Jika `folder archive` tidak ada, maka `folder submissions` akan terkena `rename` menjadi `archive`. Perintah `mv` bisa digunakan untuk melakukan `rename`.
- Kita juga bisa memindahkan direktori dari mana saja jika mengetahui lokasi persis direktorinya, seperti `mv /Projects/Work /Archives`.
- Untuk melakukan `rename file` dan langsung memindahkannya ke direktori lain dalam satu jalan: `mv /Documents/notes.txt /Markdown/notes.md`.

2.20 Tips Command Line

- Gunakan perintah `cd` untuk bernavigasi antar direktori.
- Gunakan `pwd` untuk mengonfirmasi direktori kerja Anda saat ini.
- Ingatlah bahwa `files` dan `directories` bersifat `case-sensitive` di `command line`.
- Direktori biasanya muncul dengan `d` di awal ketika menggunakan `ls -l`.

2.21 Perintah wc (Word Count)

- Perintah `wc` digunakan untuk menghitung jumlah kata dalam file. Misalnya, `wc -w file1.txt` memberikan jumlah kata.
- `Flag -w` untuk menghitung jumlah kata dan `flag -c` untuk menghitung jumlah karakter.

2.22 Pipe

- Menggunakan `pipes` (`|`), Anda dapat mengalirkan `output` dari satu perintah ke perintah lain. Contohnya, `cat file1.txt | wc -w` untuk menghitung kata dalam `File1.txt`.

2.23 Redirection dalam Linux

2.23.1 Dasar Redirection

- `Standard Input (stdin)`: Input standar berasal dari `keyboard`. Tanda yang digunakan untuk input adalah `<`. Diwakili angka 0.

- `Standard Output (stdout)`: Output standar dari perintah seperti `ls` dikirim ke layar. Tanda yang digunakan untuk mengarahkan `output` ke file adalah `>`. Diwakili angka 1.
- `Standard Error (stderr)`: Ketika terjadi kesalahan, `output error` dikirim ke `stderr`. Tanda untuk mengarahkan `error` adalah `2>`. Diwakili angka 2.

2.23.2 Jenis Redirection dan Contoh Praktis

- `Standard Input`: `cat > input.txt` untuk menyimpan `input` pengguna. Tekan `Ctrl + D` untuk menandai akhir `input`.
- `Standard Output`: `ls -l > output.txt` untuk menyimpan hasil perintah `ls` ke dalam file `output.txt`.
- `Standard Error`: `ls -l /bin/usr 2> error.txt` untuk menyimpan pesan `error` ke dalam file `error.txt`.
- Menggabungkan `Output` dan `Error`: `ls -l /bin/usr > output.txt 2>&1` untuk mengarahkan keduanya ke `output.txt`.

2.24 Pengantar tentang Grep

- `Grep` adalah singkatan dari `global regular expression print`, yang digunakan untuk mencari di seluruh file dan folder serta konten file.

2.25 Pencarian dengan Grep

- Untuk mencari nama yang dimulai dengan "Sam", perintah yang digunakan adalah `grep Sam names.txt`, yang mengembalikan daftar nama yang sesuai.
- `Grep` bersifat `case sensitive`.

2.26 Menggunakan Flag pada Grep

- Dengan menambahkan `flag -i`, perintah `grep -i Sam names.txt` akan mengabaikan perbedaan huruf besar dan kecil.
- `Flag -w` digunakan untuk pencarian yang tepat (`exact match`), seperti `grep -w Sam names.txt`.

2.27 Menggabungkan Pencarian dengan Pipe dan Grep

- `Grep` juga dapat digunakan dengan `pipe` untuk menggabungkan pencarian, misalnya dengan perintah `ls /bin | grep zip` untuk mencari file `executable` yang mengandung "zip".

3 Working with Git

3.1 Git dan GitHub

- **Git** adalah sistem `version control` terdistribusi (`DVCS`) yang dirancang untuk melacak perubahan pada file dalam proyek.
- **Sejarah Git**: Diciptakan oleh `Linus Torvalds` pada tahun 2005 untuk mengelola `kernel Linux`.
- **Manfaat Git**: Menawarkan kecepatan dan kinerja tinggi; merupakan perangkat lunak `gratis` dan `open source`.

3.1.1 GitHub

- **Definisi**: Layanan `hosting` berbasis `cloud` yang memungkinkan pengguna mengelola `Git repositories` melalui antarmuka pengguna (`GUI`).
- **Fitur Utama**: Menyediakan `kontrol akses`, `pull requests`, dan `otomatisasi`.
- **Popularitas**: Berfungsi sebagai jaringan sosial untuk pengembang, memfasilitasi kontribusi kode global pada proyek `publik` maupun `privat`.

3.2 Dasar-Dasar Repository dan Branch

3.2.1 Repository

- **Repository**: Tempat penyimpanan proyek yang dikelola dengan Git.
- **Lokal (`Local`)**: Merujuk pada mesin Anda (laptop/desktop) yang hanya dapat diakses oleh Anda.
- **Remote**: Merupakan `repository` yang berada di `server` (*misalnya*, GitHub) yang dapat diakses oleh banyak pengembang.

3.2.2 Upstream

- **Upstream** merujuk pada `branch` di `remote repository` yang terhubung dengan `branch` lokal Anda. Ini adalah sumber dari mana Anda menarik (`pull`) pembaruan dan ke mana Anda mengirim (`push`) perubahan.
- **Fungsi**: Dapat diatur berbeda dari `branch main`. Perintah `git push -u` digunakan untuk memilih `branch remote` yang ingin dikaitkan dengan `branch` lokal.

3.2.3 Branch Main

- **Branch Main**: `Branch` utama dalam `repositori` yang biasanya menyimpan versi stabil dari kode yang siap untuk produksi.
- **Fungsi**: Tempat menggabungkan semua perubahan yang telah diuji dan disetujui dari `branch` lain.

—

3.3 Alur Kerja Git (Status File)

Alur kerja Git terdiri dari tiga status utama yang melacak perubahan file:

- **Modified**: Git mengetahui file telah berubah (ditambahkan/dihapus/diperbarui) tetapi tidak melacaknya secara eksplisit untuk `commit` berikutnya.
- **Staged**: File telah dipindahkan ke `staging area` (`index`) untuk dikomit. Ini adalah area persiapan sebelum `commit` dilakukan.
- **Committed**: Perubahan telah disimpan dan membuat `snapshot` dari kondisi `repository` saat ini. Ini berfungsi sebagai titik simpan.

3.3.1 Alur Kerja Dasar dan Perintah

- **Memeriksa Status**: Gunakan `git status` untuk memeriksa `branch` yang digunakan dan apakah ada perubahan yang perlu dikomit.
- **Menambahkan File**: Gunakan `git add <file>` untuk memindahkan file dari `working directory` ke `staging area`.
- **Melakukan Commit**: Gunakan `git commit -m "<pesan>"` untuk menyimpan perubahan yang telah di-`stage`.
- **Mengirim Perubahan**: Perubahan hanya ada di lokal sampai perintah `git push` dijalankan untuk mengunggah ke `remote repository`.

```
# Membuat file baru
touch test.txt
# Memeriksa status (menunjukkan 'untracked')
git status
# Mulai melacak file (status menjadi 'staged')
git add test.txt
# Menyimpan perubahan
git commit -m "adding␣a␣new␣file␣for␣testing"
# Mengunggah perubahan ke remote
git push
```

Listing 1: Contoh Alur Kerja Dasar Git

—

3.4 Mengelola Branch dan Pull Request

3.4.1 Membuat dan Berpindah Branch

Perintah `git checkout` memiliki dua fungsi:

- **Pindah Branch**: Mengganti ke `branch` yang sudah ada (`git checkout main`).
- **Buat dan Pindah**: Membuat `branch` baru sekaligus berpindah ke `branch` tersebut.

```
# Membuat branch baru dan langsung berpindah
git checkout -b feature/lesson
# Mengirim branch baru ke remote dan mengatur upstream
git push -u origin feature/lesson
```

Listing 2: Membuat Branch dan Berpindah

3.4.2 Workflow Pull Request (PR)

- Setelah `branch` baru dikirim (`push`), GitHub akan menyediakan opsi untuk membuat **Pull Request**.
- **PR** memungkinkan tim untuk meninjau perubahan (`peer review`) sebelum digabungkan (`merge`) ke `branch main`.
- Menggunakan `branch` terpisah untuk fitur baru adalah praktik terbaik untuk menghindari konflik.

3.4.3 Git Workflow

- **Workflow** adalah proses terstruktur yang membantu tim mengelola proyek, memandu dari `pull` hingga `push`.
- **Feature Branching**: Metode membuat `branch` baru dari `main` untuk mengerjakan fitur tertentu. `Branch main` tetap stabil hingga fitur selesai dan siap di-`merge`.
- **Praktik Terbaik**: Sebelum `git push`, selalu lakukan `git pull` untuk mengambil perubahan terbaru dari `remote` dan mengurangi kemungkinan `merge conflict`.

—

3.5 Inspeksi dan Perbandingan Kode

3.5.1 HEAD dan Struktur `.git`

- **Folder** `.git`: Menyimpan semua informasi perubahan.
- **HEAD**: Adalah `pointer` yang menunjuk ke `commit` saat ini (terbaru) pada `branch` yang aktif. Setiap kali ada `commit` baru, `ID commit` di `HEAD` diperbarui.

```
# Masuk ke folder .git
cd .git
# Melihat branch yang sedang ditunjuk HEAD
cat HEAD
```

Listing 3: Melihat Referensi HEAD

3.5.2 Git Diff

- **git diff** membantu pengguna untuk melihat perbedaan antara versi file yang berbeda (file, `branch`, atau `commit`).
- **Perbandingan Commit**: Gunakan `git log` untuk melihat `ID commit` dan `git diff <ID1> <ID2>` untuk membandingkan perubahannya.

3.5.3 Git Blame

- **git blame** digunakan untuk melihat setiap baris perubahan pada file tertentu dan menunjukkan: `ID commit`, `author`, `timestamp`, `line number`, dan `content`.
- **Penggunaan**: Berguna untuk melacak siapa yang melakukan perubahan dan kapan, terutama dalam proyek dengan banyak pengembang.

```
# Melihat siapa yang mengubah setiap baris di file
git blame file.txt
# Membatasi output dari baris 5 hingga 15
git blame -L 5,15 file.txt
```

Listing 4: Penggunaan Git Blame

—

3.6 Merge Conflict dan Forking

3.6.1 Merge Conflict

- **Merge Conflict** terjadi ketika Git tidak dapat secara otomatis menggabungkan perubahan dari dua `branch` yang berbeda karena ada perubahan yang saling bertentangan pada baris kode yang sama.

- **Penyelesaian**: Pengembang harus meninjau dan menyelesaikan konflik secara manual, lalu melakukan `git add` dan `git commit` untuk menandai konflik telah selesai.

3.6.2 Forking

- **Forking** adalah proses membuat **salinan lengkap** dari sebuah `repository` di akun GitHub pengguna sendiri.
- **Perbedaan dengan Branching**: `Branching` dilakukan dalam `repository` yang sama, sementara `forking` menciptakan `repository` yang sepenuhnya independen di bawah akun pengguna lain.
- **Kontribusi**: Setelah perubahan dilakukan di `forked repository`, pengguna membuat **Pull Request** kembali ke `repository` asli.