Lokakarya GIT

Universitas Pradita

Powered by ChatGPT

Alfa Yohannis

September 24, 2024

PRADITA University

Daftar Isi

Bab 1

Sistem Version Control dan Git

1.1 Pendahuluan

Apa itu Sistem Version Control?
Sistem version control (VCS) adalah alat yang membantu mengelola perubahan pada kode sumber atau dokumen dari waktu ke waktu. VCS memungkinkan beberapa pengembang untuk berkolaborasi, melacak perubahan, dan mengembalikan ke versi sebelumnya jika diperlukan. VCS mencatat setiap perubahan yang dilakukan pada file, sehingga memudahkan pelacakan evolusi proyek.

Beberapa manfaat utama menggunakan VCS antara lain:

- **Kolaborasi**: Beberapa anggota tim dapat bekerja pada proyek yang sama secara bersamaan tanpa saling menimpa pekerjaan satu sama lain.
- **Riwayat**: Setiap perubahan dicatat, memberikan riwayat lengkap dari modifikasi yang dilakukan.
- **Kemampuan Mengembalikan**: Dimungkinkan untuk mengembalikan ke versi sebelumnya jika terjadi kesalahan.
- **Branching dan Merging**: Fitur atau perbaikan yang berbeda dapat dikembangkan secara terpisah, lalu digabungkan kembali ke dalam proyek utama setelah selesai.

1.2 Jenis-jenis Sistem Version Control

1.2.1 Local Version Control System (LVCS)

Sistem version control lokal (Local Version Control System) menyimpan semua versi file secara lokal di komputer pengguna. Pada umumnya, salinan file dibuat secara manual di direktori yang berbeda untuk melacak perubahan. Namun, metode ini rentan terhadap kesalahan karena tidak ada mekanisme otomatis untuk melacak versi yang lebih kompleks.

Contoh produk:

- File system sederhana yang digunakan untuk membuat salinan file secara manual.

1.2.2 Centralized Version Control System (CVCS)

Centralized Version Control System (CVCS) menggunakan server pusat yang menyimpan semua versi file. Pengguna harus terhubung ke server ini untuk mendapatkan atau memperbarui versi file. Sistem ini mempermudah kolaborasi, tetapi memiliki kelemahan jika server pusat mengalami kegagalan.

Contoh produk:

- **Subversion (SVN)**: Sistem version control terpusat yang populer dan digunakan di banyak perusahaan dan proyek perangkat lunak.

- **Perforce**: Sistem VCS yang digunakan oleh perusahaan besar untuk proyek pengembangan perangkat lunak berskala besar.

- **CVS (Concurrent Versions System)**: Sistem version control yang lebih tua, namun masih digunakan di beberapa proyek.

1.2.3 Distributed Version Control System (DVCS)

Distributed Version Control System (DVCS) memungkinkan setiap pengguna memiliki salinan lengkap dari seluruh riwayat proyek di komputer masing-masing. Pengguna dapat bekerja secara offline, dan perubahan dapat digabungkan baik dengan server pusat atau dengan pengguna lain. Sistem ini menawarkan fleksibilitas dan keamanan lebih tinggi karena data disimpan di banyak tempat.

Contoh produk:

- **Git (Global Information Tracker)**: Sistem version control terdistribusi yang paling populer dan digunakan dalam banyak proyek open-source dan perusahaan besar.

- **Mercurial**: Alternatif untuk Git yang menawarkan beberapa kesederhanaan dan kemudahan penggunaan.

- **Bazaar**: Sistem VCS yang digunakan di beberapa komunitas open-source.

1.3 Latar Belakang Git

Git adalah sistem version control terdistribusi, yang awalnya dikembangkan oleh Linus Torvalds pada tahun 2005 untuk mengelola pengembangan kernel Linux. Pada saat itu, tidak ada sistem version control yang memenuhi kebutuhan kinerja, fleksibilitas, dan skalabilitas yang tinggi untuk pengembangan Linux.

Git diciptakan dengan beberapa tujuan desain utama:

- **Kecepatan**: Penanganan tugas-tugas version control seperti commit, branching, dan merging dengan cepat.

- **Desain Sederhana**: Git dirancang agar mudah digunakan namun cukup kuat untuk menangani alur kerja yang kompleks.

- **Dukungan Kuat untuk Pengembangan Non-linear**: Ini dicapai melalui kemampuan branching dan merging yang kuat.

- **Pengembangan Terdistribusi**: Setiap pengembang memiliki salinan penuh dari sejarah proyek secara lokal, sehingga dapat bekerja secara offline dan tetap memiliki akses ke seluruh riwayat versi.

Sejak diciptakan, Git telah menjadi sistem version control yang paling populer di dunia, digunakan secara luas oleh pengembang perangkat lunak, organisasi, dan komunitas open-source.

1.4 Instal Git Secara Lokal

Git adalah sistem version control terdistribusi yang banyak digunakan untuk pengembangan perangkat lunak dan tugas-tugas version control lainnya. Untuk mulai menggunakan Git, instalasi perlu dilakukan secara lokal di komputer.

1.4.1 Menginstal Git di Windows

- Unduh installer Git dari situs resmi: `https://git-scm.com/download/win`
- Jalankan installer dan ikuti petunjuknya. Pilih opsi default kecuali jika ada kebutuhan khusus.
- Setelah instalasi selesai, buka command prompt dan ketik `git --version` untuk memverifikasi bahwa Git telah diinstal dengan sukses.

```
git --version
```

1.4.2 Menginstal Git di macOS

- Buka terminal dan jalankan perintah: `xcode-select --install`. Perintah ini akan menginstal Git dan alat pengembang command line dari Apple.

```
xcode-select --install
```

- Sebagai alternatif, gunakan package manager seperti Homebrew dengan menjalankan: `brew install git`.

```
brew install git
```

- Verifikasi instalasi dengan mengetik `git --version`.

```
git --version
```

1.4.3 Menginstal Git di Linux

- Pada distribusi berbasis Debian seperti Ubuntu, jalankan: `sudo apt-get install git`.

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

- Pada distribusi berbasis Red Hat, jalankan: `sudo yum install git`.

```
sudo yum install git
```

- Verifikasi instalasi dengan `git --version`.

```
git --version
```

1.5 Membuat Akun GitHub

GitHub adalah platform populer untuk menghosting repository Git di cloud. GitHub menawarkan rencana gratis dan berbayar untuk mengelola repository.

1. Buka `https://github.com/` dan klik tombol `Sign up`.
2. Isi detail yang diperlukan, termasuk alamat email yang valid dan nama pengguna yang unik.
3. Verifikasi email dan selesaikan proses pembuatan akun.
4. Setelah masuk, mulai membuat repository dan berkolaborasi dengan yang lain.

Bab 2

Penggunaan Git Dasar

2.1 Git Init: Menginisialisasi Repository

Perintah `git init` digunakan untuk menginisialisasi repository Git baru. Perintah ini akan membuat direktori `.git` di dalam folder proyek, yang akan menyimpan semua metadata dan riwayat version control.

```
git init
```

Setelah perintah ini dijalankan, folder proyek sekarang menjadi repository Git dan siap untuk melacak perubahan.

2.2 Git Add: Menambahkan Perubahan ke Staging

Perintah `git add` digunakan untuk menambahkan perubahan pada file ke area staging, yaitu tahap sebelum perubahan tersebut dikomit ke repository. File individu atau semua file yang telah diubah bisa ditambahkan sekaligus.

```
git add <nama_file>
git add .
```

Menggunakan `git add .` akan menambahkan semua file yang berubah ke staging.

2.3 Git Commit: Mengkomit Perubahan

Setelah perubahan ditambahkan ke area staging, perintah `git commit` digunakan untuk mengunci perubahan ke dalam repository. Setiap `commit` memerlukan pesan yang menjelaskan perubahan yang dilakukan.

```
git commit -m "Deskripsi perubahan"
```

Pesan commit harus jelas dan menjelaskan apa yang telah diubah untuk memudahkan pelacakan riwayat perubahan.

2.4 Git Reset: Menghapus Perubahan dari Staging

Jika ingin membatalkan perubahan yang telah ditambahkan ke staging tanpa mengubah file yang sebenarnya, gunakan perintah `git reset`.

```
git reset <nama_file>
```

Perintah ini akan menghapus file dari staging, namun perubahan pada file tetap ada. Untuk membatalkan perubahan pada file itu sendiri, gunakan `git checkout`.

2.5 Penggunaan File .gitignore

File `.gitignore` digunakan untuk menentukan file atau direktori mana yang tidak ingin dilacak di dalam repository Git. Pola atau nama file yang tidak akan ditambahkan ke staging meskipun diubah bisa ditentukan.

```
# Contoh file .gitignore
node_modules/
*.log
*.tmp
```

Dengan menambahkan pola file ke `.gitignore`, Git akan mengabaikan file-file tersebut.

2.6 Membuat dan Menggabungkan Branch

Git memungkinkan pekerjaan pada berbagai fitur atau perbaikan secara terpisah menggunakan *branch*. Perintah `git branch` digunakan untuk membuat cabang baru, dan `git checkout` untuk berpindah ke cabang tersebut.

```
git branch <nama_branch>
git checkout <nama_branch>
```

Setelah perubahan selesai, cabang tersebut bisa digabungkan ke dalam cabang utama (`main` atau `master`) menggunakan perintah `git merge`.

```
git checkout main
git merge <nama_branch>
```

Proses ini akan menggabungkan perubahan yang dilakukan pada cabang lain ke dalam cabang utama.

Bab 3

Menggunakan Git dengan GitHub

3.1 Membuat Repository Private/Public

GitHub memungkinkan pembuatan repository baik private maupun public untuk mengelola proyek. Berikut adalah langkah-langkah untuk membuat repository di GitHub:

1. Masuk ke akun GitHub dan klik tombol `New Repository` di halaman utama atau navigasi profil.
2. Berikan nama repository dan deskripsi opsional.
3. Pilih apakah repository tersebut akan bersifat `Public` (dapat diakses oleh siapa saja) atau `Private` (hanya bisa diakses oleh pemilik dan kolaborator yang diizinkan).
4. Pilih apakah ingin menambahkan file `README.md`, `.gitignore`, atau lisensi.
5. Klik `Create repository`.

Setelah repository dibuat, URL Git untuk repository tersebut akan diperoleh, yang dapat digunakan sebagai remote di proyek lokal.

3.2 Menambahkan Remote dan Push Perubahan

Setelah repository di GitHub dibuat, langkah selanjutnya adalah menambahkan repository tersebut sebagai *remote* di proyek Git lokal. Berikut langkah-langkahnya:

3.2.1 Menambahkan Remote

Untuk menambahkan remote ke repository lokal, gunakan perintah `git remote add` dengan URL GitHub dari repository.

```
1  git remote add origin https://github.com/username/nama-repo.git
```

Pada perintah ini, `origin` adalah nama default untuk remote repository utama. Nama lain dapat digunakan jika diinginkan.

3.2.2 Push Perubahan ke Remote

Setelah remote ditambahkan, perubahan dari repository lokal dapat dikirim ke repository di GitHub menggunakan perintah `git push`.

```
1  git push -u origin main
```

Perintah ini akan mengirim semua commit di cabang `main` ke remote `origin` (GitHub). Opsi `-u` digunakan untuk mengatur `main` sebagai cabang default untuk operasi `push` berikutnya.

3.2.3 Sinkronisasi Perubahan

Untuk menarik (pull) perubahan dari repository remote ke lokal, gunakan perintah:

```
1  git pull origin main
```

Perintah ini memastikan repository lokal tetap sinkron dengan repository GitHub.

Bab 4

Belajar Bekerja Kolaboratif

Dalam proyek kolaboratif, penting untuk mengetahui cara bekerja dengan orang lain secara efektif menggunakan Git dan GitHub. Bab ini mencakup konsep dan praktik kunci untuk kolaborasi, termasuk memfork repository, mengkloning repository, membuat pull request, dan mengelola penggabungan.

4.1 Memfork Repository

Memfork sebuah repository memungkinkan pengguna untuk membuat salinan pribadi dari proyek orang lain. Hal ini memungkinkan eksperimen dan modifikasi tanpa mempengaruhi proyek asli. Untuk memfork sebuah repository:

1. Arahkan ke repository GitHub yang ingin difork.
2. Klik tombol `Fork` di bagian kanan atas halaman.
3. Pilih akun sebagai tujuan untuk repository yang difork.

Setelah repository difork, salinan baru dibuat di bawah akun GitHub, yang dapat dikloning dan dimodifikasi secara independen.

4.2 Mengkloning Repository

Mengkloning repository memungkinkan pengguna untuk mengunduh salinan repository ke mesin lokal. Ini dapat dilakukan menggunakan perintah berikut di terminal:

```
1  git clone <repository-url>
```

Gantilah `<repository-url>` dengan URL dari repository yang difork atau asli. Perintah ini membuat salinan lokal, memungkinkan pengguna bekerja secara offline.

4.3 Membuat Pull Request

Setelah melakukan perubahan pada repository yang dikloning, langkah berikutnya adalah mengirimkan perubahan tersebut untuk ditinjau. Ini dilakukan dengan membuat pull request:

1. Dorong perubahan ke repository yang difork menggunakan:

```
1     git push origin <branch-name>
```

2. Buka repository asli di GitHub.
3. Klik pada tab `Pull requests`.

4. Klik tombol `New pull request`.
5. Pilih cabang dan berikan deskripsi tentang perubahan yang dilakukan.
6. Klik `Create pull request`.

Pull request memfasilitasi diskusi dan tinjauan perubahan sebelum digabungkan ke dalam proyek utama.

4.4 Mengambil dan Mengirim Perubahan di GitHub

Untuk menyinkronkan perubahan antara repository lokal dan repository jarak jauh di GitHub, gunakan perintah berikut:

- Untuk mengambil perubahan terbaru dari repository jarak jauh:

```
git pull origin <branch-name>
```

- Untuk mengirim perubahan lokal ke repository jarak jauh:

```
git push origin <branch-name>
```

Perintah ini memastikan bahwa semua kontributor bekerja dengan versi kode yang paling mutakhir.

4.5 Menyelesaikan Konflik Penggabungan

Ketika beberapa kontributor melakukan perubahan pada bagian yang sama dari sebuah file, konflik penggabungan dapat terjadi. Untuk menyelesaikan konflik penggabungan:

1. Ambil perubahan terbaru dari repository jarak jauh.
2. Jika terjadi konflik, Git akan menandai file yang konflik.
3. Buka file yang konflik dan cari penanda konflik (misalnya, `<<<<<<< HEAD`).
4. Edit file secara manual untuk menyelesaikan konflik.
5. Setelah menyelesaikan konflik, siapkan perubahan:

```
git add <resolved-file>
```

6. Komit perubahan yang telah diselesaikan:

```
git commit -m "Resolved merge conflict"
```

Dengan mengikuti langkah-langkah ini, kontributor dapat mengelola dan menyelesaikan konflik penggabungan secara efektif dalam proyek kolaboratif.

Bab 5

Praktik Terbaik dalam Menggunakan GitHub

Menggunakan GitHub secara efektif tidak hanya melibatkan penguasaan perintah Git, tetapi juga menerapkan praktik terbaik untuk menjaga repository tetap bersih dan terorganisir. Berikut adalah beberapa praktik terbaik yang disarankan:

5.1 Hindari Mengunggah File Binary Target

File binary target, seperti file hasil kompilasi atau file executable, sebaiknya tidak diunggah ke repository. Ini karena file tersebut dapat dengan cepat membesar dan tidak perlu dilacak di dalam version control. Sebagai gantinya, cukup simpan kode sumbernya dan biarkan setiap pengguna melakukan kompilasi pada mesin masing-masing.

5.2 Hindari File Binary Berukuran Besar

File binary yang berukuran besar, seperti gambar, video, atau file media lainnya, sebaiknya dihindari dalam repository Git. Gunakan platform penyimpanan eksternal seperti Git LFS (Large File Storage) atau layanan penyimpanan awan untuk menyimpan file-file tersebut. Ini akan membantu menjaga ukuran repository tetap kecil dan mudah dikelola.

5.3 Hindari File Temporary

File sementara yang dihasilkan selama proses pengembangan, seperti file log, file cache, atau file konfigurasi lokal, sebaiknya tidak diunggah ke repository. Gunakan file `.gitignore` untuk mengabaikan file-file ini agar tidak termasuk dalam version control.

```
# Contoh isi .gitignore
*.log
*.tmp
*.cache
```

5.4 Tulis Pesan *Commit* yang Jelas

Pesan commit yang jelas dan deskriptif sangat penting untuk memudahkan pengembang lain memahami perubahan yang dilakukan. Gunakan kalimat yang menjelaskan apa yang diubah dan mengapa. Contoh:

```
git commit -m "Memperbaiki bug pada fungsi login dan menambahkan validasi
    input"
```

5.5 Gunakan Branch untuk Fitur Baru dan Perbaikan

Selalu buat branch baru untuk mengembangkan fitur baru atau melakukan perbaikan. Ini akan menjaga cabang utama tetap stabil dan menghindari konflik yang tidak perlu.

```
1  git checkout -b nama_fitur_baru
```

5.6 Jaga *Commit* yang Kecil dan Sering

Commit yang kecil dan sering lebih mudah dikelola dan ditelusuri daripada komitmen yang besar dan jarang. Hal ini juga memudahkan pengembalian perubahan jika terjadi kesalahan.

5.7 Perbarui `README.md` Secara Berkala

File `README.md` merupakan dokumen penting yang menjelaskan proyek. Pastikan informasi di dalamnya selalu diperbarui, termasuk petunjuk instalasi, penggunaan, dan dokumentasi lainnya.

5.8 Gunakan Pull Request untuk Kolaborasi

Saat bekerja dalam tim, gunakan fitur pull request untuk mengusulkan perubahan. Ini memfasilitasi diskusi dan tinjauan sebelum perubahan diterapkan ke cabang utama.

5.9 Bersihkan Repository Secara Berkala

Lakukan pemeriksaan rutin pada repository untuk menghapus file yang tidak diperlukan dan memperbaiki masalah. Ini termasuk menghapus branch yang sudah tidak digunakan dan memperbarui file `.gitignore`.

5.10 Patuhi Konvensi Penamaan

Gunakan konvensi penamaan yang konsisten untuk branch, commit, dan file. Ini akan memudahkan pengembang lain memahami struktur dan tujuan dari setiap elemen di dalam repository.

Dengan mengikuti praktik terbaik ini, penggunaan GitHub akan lebih efektif dan kolaborasi dalam proyek dapat berjalan lebih lancar.

Bab 6

Reset dan Rollback dalam Git

Dalam pengembangan perangkat lunak, ada kalanya kita perlu membatalkan atau mengganti perubahan yang telah dilakukan di dalam repository Git. Git menyediakan beberapa perintah yang memungkinkan pengguna untuk melakukan reset atau rollback terhadap perubahan yang telah dilakukan, baik di area kerja (working directory) maupun dalam riwayat commit.

6.1 Menggunakan `git reset`

Perintah `git reset` digunakan untuk memindahkan posisi `HEAD` dan dapat digunakan untuk membatalkan perubahan di area staging atau menghapus commit dari riwayat. Ada tiga mode utama dalam `git reset`:

1. `git reset --soft <commit>`: Memindahkan `HEAD` ke commit yang ditentukan, tetapi tidak mengubah area staging atau area kerja. Commit tetap ada di area staging.
2. `git reset --mixed <commit>`: Mode default, memindahkan `HEAD` ke commit yang ditentukan dan menghapus perubahan dari area staging, tetapi tidak dari area kerja.
3. `git reset --hard <commit>`: Memindahkan `HEAD` ke commit yang ditentukan dan menghapus perubahan dari area staging serta area kerja. Semua perubahan yang belum di-commit akan hilang.

6.1.1 Contoh Penggunaan `git reset`

```
# Melakukan reset soft ke commit sebelumnya
git reset --soft HEAD~1

# Melakukan reset mixed ke commit dengan hash tertentu
git reset --mixed abc1234

# Melakukan reset hard ke commit sebelumnya, menghapus perubahan di area kerja
git reset --hard HEAD~1
```

Reset sangat berguna ketika Anda ingin membatalkan commit terakhir atau kembali ke status sebelumnya tanpa meninggalkan riwayat commit.

6.2 Menggunakan `git reset` untuk File Tertentu

Git juga memungkinkan melakukan reset untuk file tertentu tanpa mempengaruhi keseluruhan commit. Ini berguna jika Anda ingin membatalkan perubahan pada satu file tanpa mempengaruhi file lainnya.

6.2.1 Contoh Penggunaan `git reset` untuk File Tertentu

```
# Menghapus file dari area staging dan mengembalikannya ke kondisi sebelum di
    -commit
git reset HEAD <nama-file>
```

Perintah ini akan menghapus file dari staging area dan mengembalikan file ke kondisi sebelumnya tanpa mengubah file lain di staging.

6.3 Menggunakan `git checkout` untuk Rollback File Tertentu

Selain menggunakan `git reset`, Anda juga dapat menggunakan `git checkout` untuk mengembalikan file tertentu ke kondisi commit tertentu atau commit terakhir. Perintah ini mengembalikan file yang dipilih ke status yang ada di commit tertentu.

6.3.1 Contoh Penggunaan `git checkout` untuk Rollback File Tertentu

```
# Mengembalikan file ke kondisi commit sebelumnya
git checkout HEAD~1 -- <nama-file>

# Mengembalikan file ke kondisi commit tertentu
git checkout abc1234 -- <nama-file>
```

Dengan `git checkout`, Anda bisa mengembalikan file ke versi commit yang telah disimpan sebelumnya tanpa mempengaruhi perubahan file lain.

6.4 Menggunakan `git revert`

`git revert` adalah perintah yang digunakan untuk membatalkan perubahan dari commit sebelumnya dengan cara membuat commit baru yang membalikkan perubahan tersebut. Berbeda dengan `git reset`, `git revert` tidak mengubah riwayat commit dan lebih cocok digunakan ketika bekerja dalam tim.

6.4.1 Contoh Penggunaan `git revert`

```
# Membatalkan commit terakhir dengan membuat commit baru
git revert HEAD

# Membatalkan commit tertentu dengan hash
git revert abc1234
```

Setelah menjalankan perintah ini, Git akan membuat commit baru yang berisi kebalikan dari perubahan yang ada di commit yang di-revert, sehingga riwayat commit tetap utuh dan aman untuk kolaborasi.

6.5 Menggunakan `git checkout`

Perintah `git checkout` dapat digunakan untuk mengembalikan file tertentu ke kondisi seperti di commit sebelumnya atau pindah ke branch/cabang yang lain. Untuk mengembalikan file ke kondisi commit sebelumnya:

```
# Mengembalikan file ke kondisi di commit sebelumnya
git checkout HEAD~1 -- <nama-file>

```

```
# Mengembalikan semua file ke kondisi di commit tertentu
git checkout abc1234 -- <nama-file>
```

Perintah ini berguna ketika Anda hanya ingin membatalkan perubahan pada beberapa file tanpa mempengaruhi seluruh commit atau branch.

6.6 Menggunakan `git reflog`

`git reflog` adalah fitur yang mencatat setiap perubahan pada posisi `HEAD`, memungkinkan Anda untuk melihat riwayat posisi `HEAD`, termasuk commit yang telah di-reset atau dihapus. Ini sangat berguna jika Anda secara tidak sengaja melakukan `reset --hard` dan ingin memulihkan commit yang hilang.

6.6.1 Contoh Penggunaan `git reflog`

```
# Melihat riwayat posisi HEAD
git reflog

# Memulihkan commit yang hilang dengan git reset ke posisi HEAD sebelumnya
git reset --hard HEAD@{2}
```

Dengan `git reflog`, Anda dapat mengembalikan repository ke kondisi sebelumnya meskipun commit telah dihapus atau di-reset.

6.7 Tips untuk Menggunakan Reset dan Rollback dengan Aman

- Gunakan `git reset --soft` atau `--mixed` jika Anda ingin menghindari kehilangan perubahan di area kerja.
- Selalu cek status repository sebelum menggunakan `git reset --hard` untuk memastikan Anda tidak menghapus perubahan yang penting.
- Lebih baik menggunakan `git revert` daripada `git reset` saat bekerja dalam tim untuk menjaga integritas riwayat commit.
- Gunakan `git reflog` untuk memulihkan commit yang hilang setelah melakukan reset atau operasi berbahaya lainnya.

Dengan menggunakan `git reset`, `git revert`, dan `git reflog` dengan bijak, Anda dapat mengelola perubahan di dalam repository Git dengan lebih fleksibel dan aman.

Bab 7

Latihan Resolusi Conflict

7.1 Menginstal Git di Linux

- Pada distribusi berbasis Debian seperti Ubuntu, jalankan: `sudo apt-get install git`.

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

- Verifikasi instalasi dengan `git --version`.

```
git --version
```

7.2 Registrasi ke GitHub

Untuk menggunakan GitHub, langkah pertama adalah mendaftar untuk akun. Berikut adalah langkah-langkah untuk mendaftar ke GitHub:

1. Buka situs web GitHub di `https://github.com`.
2. Klik tombol `Sign up` di pojok kanan atas halaman.
3. Masukkan informasi yang diperlukan, termasuk alamat email, username, dan password.
4. Pilih paket yang diinginkan, bisa memilih antara `Free` atau berlangganan untuk opsi yang lebih banyak.
5. Klik `Create account` setelah mengisi informasi.
6. Ikuti petunjuk yang diberikan untuk memverifikasi akun melalui email.
7. Setelah verifikasi, masuk ke akun GitHub yang baru dibuat.

7.3 Instalasi GitHub CLI di Ubuntu

GitHub CLI (`gh`) adalah alat baris perintah yang memungkinkan pengguna untuk berinteraksi dengan GitHub secara langsung dari terminal. Dengan `gh`, pengguna dapat melakukan berbagai tugas seperti mengelola repositori, mengirim permintaan tarik, dan berinteraksi dengan masalah tanpa perlu menggunakan antarmuka grafis.

7.3.1 Langkah 1: Perbarui Daftar Paket

Buka terminal dan jalankan perintah berikut untuk memperbarui daftar paket:

```
sudo apt update
```

7.3.2 Langkah 2: Instal `gh` Menggunakan Pengelola Paket

Install GitHub CLI menggunakan pengelola paket `apt`:

```
sudo apt install gh
```

7.3.3 Langkah 3: Verifikasi Instalasi

Setelah instalasi selesai, verifikasi bahwa `gh` terinstal dengan benar dengan memeriksa versinya:

```
gh --version
```

7.3.4 Opsi: Autentikasi

Untuk mulai menggunakan `gh`, authentikasi dengan akun GitHub pengguna:

```
gh auth login
```

Perintah berikut menuntun dalam proses autentikasi, memungkinkan pengguna memilih metode autentikasi yang diinginkan (browser atau token).

7.3.5 Metode Alternatif: Instal dari Rilis GitHub

Jika menggunakan versi terbaru dari rilis GitHub, ikuti langkah-langkah berikut:

1. **Unduh Rilis Terbaru**:

   Kunjungi halaman *GitHub CLI releases* untuk menemukan versi terbaru dan mengunduhnya. Sebagai alternatif, pengguna dapat menggunakan `wget` atau `curl`:

   ```
   wget https://github.com/cli/cli/releases/latest/download/gh_$(
       lsb_release -cs)_amd64.deb
   ```

2. **Instal Paket**:

   ```
   sudo dpkg -i gh_*.deb
   ```

3. **Perbaiki Ketergantungan** (jika diperlukan):

   Jika Anda mengalami masalah ketergantungan, Anda dapat memperbaikinya dengan menjalankan:

   ```
   sudo apt install -f
   ```

Setelah mengikuti langkah-langkah ini, `gh` harus berhasil diinstal di sistem Ubuntu Anda.

7.4 Cara Mengunggah Proyek Baru ke GitHub

7.4.1 1. Buat Repository Baru di GitHub

- Masuk ke akun GitHub dan klik ikon `+` di sudut kanan atas, lalu pilih `New repository`.
- Isi nama repository, deskripsi (opsional), dan pilih apakah repository tersebut bersifat publik atau privat.
- Klik `Create repository`.

7.4.2 2. Inisialisasi Proyek Secara Lokal

Buka terminal dan navigasikan ke direktori proyek Anda. Jalankan perintah berikut untuk menginisialisasi repository Git baru:

```
cd /path/to/your/project
git init
```

7.4.3 3. Tambahkan File ke Repository

Tambahkan file yang ingin disertakan dalam commit pertama:

```
git add .
```

7.4.4 4. Komit Perubahan

Buat commit pertama dengan pesan deskriptif:

```
git commit -m "Initial commit"
```

7.4.5 5. Hubungkan Repository Lokal ke GitHub

Tambahkan URL repository jarak jauh yang Anda buat di GitHub. Gantilah `<USERNAME>` dengan nama pengguna GitHub Anda dan `<REPO>` dengan nama repository:

```
git remote add origin https://github.com/<USERNAME>/<NAMA-REPOSITORY>.git
```

7.4.6 6. Unggah Perubahan ke GitHub

Akhirnya, unggah perubahan lokal Anda ke repository GitHub:

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

(Gantilah `main` dengan `master` jika cabang default Anda adalah `master`).

7.5 Membuat Repository Baru dan Mengirim Proyek ke GitHub

Sebelum mengirim proyek ke GitHub, pastikan bahwa perintah `gh` sudah terinstal di sistem dan telah melakukan autentikasi ke GitHub. Berikut langkah-langkah yang perlu diikuti:

7.5.1 Persiapan Menggunakan `gh`

1. Pastikan `gh` sudah terinstal di sistem. Jika belum, instal dengan menjalankan perintah berikut untuk Ubuntu:

   ```
   sudo apt install gh
   ```

2. Setelah diinstal, lakukan autentikasi ke GitHub dengan menggunakan perintah:

   ```
   gh auth login
   ```

   Ikuti langkah-langkah autentikasi yang diberikan.

3. Untuk membuat repository baru, jalankan perintah berikut:

   ```
   gh repo create nama-repository --public
   ```

   Perintah ini akan membuat repository baru dengan nama `nama-repository` yang bersifat publik. Jika ingin membuat repository private, gunakan opsi `--private`.

7.5.2 Membuat Repository Baru dan Mengirim Perubahan ke GitHub

Jika Anda ingin membuat repository baru dan langsung mengirim proyek ke GitHub, gunakan langkah berikut:

1. Buat file `README.md`:

```
echo "# nama-proyek" >> README.md
```

2. Inisialisasi repository Git:

```
git init
```

3. Tambahkan file `README.md` ke staging area:

```
git add README.md
```

4. Commit file tersebut dengan pesan `"first commit"`:

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

5. Ganti branch default menjadi `main`:

```
git branch -M main
```

6. Tambahkan repository remote GitHub:

```
git remote add origin https://github.com/<username>/<nama-proyek>.git
```

7. Kirim (push) commit ke repository GitHub:

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

7.5.3 Mengirim Proyek yang Sudah Ada ke Repository GitHub

Jika Anda memiliki repository Git yang sudah ada dan ingin mengirimnya ke GitHub, gunakan perintah berikut:

1. Tambahkan repository remote GitHub:

```
git remote add origin https://github.com/<username>/<nama-proyek>.git
```

2. Ganti branch default menjadi `main`:

```
git branch -M main
```

3. Kirim (push) commit yang ada ke GitHub:

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

Dengan langkah-langkah di atas, Anda dapat membuat repository baru dari proyek lokal atau mengirim proyek yang sudah ada ke GitHub. Perintah `git push` akan mengirimkan commit yang ada ke repository remote di GitHub, sedangkan `-u` akan mengatur cabang `main` sebagai cabang default untuk operasi `push` di masa mendatang.

7.6 Instalasi Maven dan Membuat Proyek Java *Hello World* Menggunakan Maven

Maven adalah alat otomatisasi proyek yang sering digunakan untuk proyek Java. Maven memudahkan manajemen dependensi dan pembuatan proyek.

7.6.1 Instalasi Maven

Untuk menginstal Maven di Ubuntu, ikuti langkah-langkah berikut:

1. Update paket sistem terlebih dahulu:

```
sudo apt update
```

2. Instal Maven dengan perintah berikut:

```
sudo apt install maven
```

3. Pastikan instalasi berhasil dengan memeriksa versi Maven:

```
mvn -version
```

   Anda akan melihat informasi versi Maven yang terinstal jika instalasi berhasil.

7.6.2 Membuat Proyek Java *Hello World* Menggunakan Maven

Setelah Maven diinstal, Anda dapat membuat proyek Java sederhana. Berikut adalah langkah-langkah untuk membuat proyek *Hello World* menggunakan Maven:

1. Buat proyek Maven baru dengan perintah berikut:

```
mvn archetype:generate -DgroupId=com.example -DartifactId=hello-world
    -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode
    =false
```

   Perintah ini akan membuat proyek Java dasar dengan struktur Maven. `groupId` digunakan untuk mengidentifikasi paket proyek Anda, dan `artifactId` adalah nama proyek Anda (dalam hal ini `hello-world`).

2. Masuk ke direktori proyek yang baru saja dibuat:

```
cd hello-world
```

3. Buka file `pom.xml` dan tambahkan properti berikut untuk menggunakan Java 17 sebagai versi kompilasi ke dalam tag `<project></project>`:

```
<project>
  ...
  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
  </properties>
  ...
</project>
```

   Properti ini memastikan proyek dikompilasi menggunakan Java 17.

4. Edit file `App.java` di direktori `src/main/java/com/example/App.java`. Pastikan kode berikut ada di dalam file tersebut:

```
    package com.example;

    public class App {
      public static void main(String[] args) {
        System.out.println("Hello World!");
      }
    }
```

5. Setelah kode selesai, Anda dapat membangun proyek dengan menjalankan perintah:

```
    mvn package
```

   Maven akan mengompilasi proyek dan menghasilkan file `JAR` di direktori `target`.

6. Jalankan proyek dengan perintah berikut:

```
    java -cp target/hello-world-1.0-SNAPSHOT.jar com.example.App
```

   Anda akan melihat keluaran `Hello World!` di terminal.

Dengan mengikuti langkah-langkah ini, Anda dapat membuat dan menjalankan proyek Java sederhana menggunakan Maven dengan konfigurasi untuk Java 16.

7.7 Mengabaikan File yang Tidak Perlu dengan `.gitignore`

File `.gitignore` digunakan untuk mengabaikan file yang tidak perlu dilacak oleh Git, seperti file hasil kompilasi, file sementara, dan file log. Untuk Java, Anda bisa menggunakan template `.gitignore` yang sudah tersedia dari GitHub.

Anda dapat menemukan template `.gitignore` untuk berbagai bahasa pemrograman di:

- https://github.com/github/gitignore/tree/main

Untuk Java, cukup salin isi dari template `.gitignore` khusus Java yang tersedia di:

- https://github.com/github/gitignore/blob/main/Java.gitignore

Salin isi file tersebut dan tempel ke file `.gitignore` lokal proyek Anda. Ini akan membantu mengabaikan file dan direktori yang tidak diperlukan untuk proyek Java, seperti direktori `target/`, file `.class`, dan sebagainya.

Berikut contoh beberapa entri yang umum digunakan di `.gitignore` untuk proyek Java:

```
# File .gitignore
target/
*.class
*.log
*.tmp
*.cache
*.jar
```

Dengan menggunakan file `.gitignore` yang sesuai, Anda dapat memastikan bahwa hanya file penting yang dilacak oleh Git dan menghindari mengunggah file yang tidak relevan ke repository.

7.8 Meng-`add`, `commit`, dan `push` Proyek ke Repository Git

Setelah proyek Java *Hello World* dibuat dan berhasil dijalankan menggunakan Maven, langkah selanjutnya adalah menambahkan proyek tersebut ke repository Git, melakukan `commit` awal, dan mengirimkan proyek tersebut ke repository GitHub. Berikut adalah langkah-langkahnya:

1. Inisialisasi repository Git di direktori proyek:

```
git init
```

Perintah ini akan membuat repository Git lokal di dalam direktori proyek.

2. Tambahkan semua file proyek ke staging area menggunakan perintah:

```
git add .
```

Perintah ini menambahkan semua file dan direktori yang ada di dalam proyek ke dalam staging area, siap untuk di-commit.

3. Lakukan `commit` awal untuk menyimpan perubahan yang telah ditambahkan:

```
git commit -m "Initial commit for Maven Hello World project"
```

Pesan commit ini menjelaskan bahwa ini adalah commit awal untuk proyek Maven *Hello World*.

4. Tambahkan remote repository GitHub:

```
git remote add origin https://github.com/<nama-user>/<nama-repository
    >.git
```

Perintah ini menghubungkan repository lokal dengan repository GitHub.

5. Kirim (push) commit awal ke repository GitHub:

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

Perintah ini akan mengirimkan commit ke cabang `main` di repository GitHub dan mengatur cabang `main` sebagai default untuk `push` berikutnya.

7.9 Menambahkan Kolaborator di GitHub

Anda dapat menambahkan kolaborator dengan langkah-langkah berikut:

1. Buka situs `https://github.com` dan login ke akun GitHub Anda.
2. Buka repository yang ingin Anda tambahkan kolaboratornya.
3. Klik pada tab `Settings` di halaman repository tersebut.
4. Gulir ke bawah hingga Anda menemukan bagian `Collaborators` atau `Manage Access`. Klik tombol tersebut untuk membuka pengaturan akses kolaborator.
5. Klik tombol `Add people`.
6. Masukkan username GitHub kolaborator yang ingin Anda tambahkan.
7. Setelah menemukan pengguna yang tepat, klik tombol `Add` untuk menambahkan mereka sebagai kolaborator.
8. Kolaborator akan menerima undangan untuk berkontribusi pada repository melalui email atau notifikasi GitHub. Mereka harus menyetujui undangan tersebut sebelum bisa berkontribusi.

Dengan menggunakan metode di atas, Anda dapat dengan mudah menambahkan kolaborator ke repository GitHub untuk bekerja secara bersama-sama dalam proyek.

7.10 Meng-clone dan Memperbarui Proyek dari Repository GitHub Menggunakan Acccount Collaborator

Setelah repository GitHub telah dibuat dan mungkin telah ada beberapa kolaborator, langkah selanjutnya adalah meng-clone proyek tersebut ke mesin lokal dengan menggunakan User GitHub lain. Berikut adalah langkah-langkah yang perlu diikuti:

1. Dapatkan URL dari repository yang ingin di-clone. Anda dapat menemukan URL ini di halaman utama repository di GitHub, biasanya terletak di sebelah tombol `Code`. URL tersebut bisa berupa HTTPS atau SSH.
2. Buka terminal di mesin lokal Anda.
3. Jalankan perintah berikut untuk meng-clone repository:

```
git clone https://github.com/<nama-user>/<nama-repository>.git
```

   Gantilah `<repository-url>` dengan URL yang Anda dapatkan sebelumnya.

4. Setelah perintah ini dijalankan, Git akan membuat salinan lokal dari repository di direktori saat ini. Anda akan melihat folder baru dengan nama repository.
5. Masuk ke direktori proyek yang telah di-clone:

```
cd <nama-repository>
```

   Gantilah `<nama-repository>` dengan nama folder yang sesuai.

6. Sekarang Anda dapat mulai bekerja dengan kode. Misalnya, perbarui kode berikut:

```
package com.example;

public class App {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

   Menjadi:

```
package com.example;

public class App {
  public static void main(String[] args) {
    System.out.println("AAAA Hello World!");
  }
}
```

7. Setelah melakukan perubahan, tambahkan file yang telah diubah ke staging area dengan perintah:

```
git add .
```

8. Lakukan commit untuk menyimpan perubahan dengan pesan yang sesuai:

```
git commit -m "Update Hello World message"
```

9. Terakhir, push perubahan ke repository GitHub:

```
git push origin main
```

Dengan langkah-langkah ini, pengguna lain dapat dengan mudah meng-clone proyek dari repository GitHub, melakukan perubahan, dan berkontribusi terhadap pengembangan proyek tersebut.

7.11 Mengupdate Kode Java dan Menangani Konflik

Pada bagian ini, kode Java akan diperbarui dari versi sebelumnya. Pengguna pertama akan melakukan perubahan dan mencoba untuk melakukan push, namun akan mengalami konflik karena perubahan yang dilakukan oleh pengguna lain di repository.

7.11.1 Kode Awal

Berikut adalah kode awal yang mencetak ”Hello World!” ke konsol:

```
package com.example;

public class App {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

7.11.2 Mengupdate Kode

Kode di atas akan diperbarui menjadi:

```
package com.example;
public class App {
  public static void main(String[] args) {
    System.out.println("XXXX Hello World!");
  }
}
```

Setelah melakukan pembaruan, lakukan langkah-langkah berikut untuk menambahkan, melakukan commit, dan push:

1. Tambahkan perubahan ke staging area:

```
    git add .
```

2. Lakukan commit dengan pesan deskriptif:

```
    git commit -m "Update Hello World message to XXXX"
```

3. Coba lakukan push ke repository GitHub:

```
    git push origin main
```

7.11.3 Error saat Push

Ketika mencoba untuk melakukan push, mungkin akan muncul pesan error seperti berikut:

```
! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/<username>/<nama-
    repository>.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
```

Pesan error ini menunjukkan bahwa branch lokal tertinggal dari branch di repository remote.

7.11.4 Mengambil Perubahan dari Remote

Untuk menyelesaikan konflik, lakukan pull untuk mengambil perubahan terbaru dari repository:

```
git pull origin main
```

7.11.5 Error saat Pull

Saat melakukan pull, akan muncul pesan error seperti berikut jika ada konflik:

```
Auto-merging App.java
CONFLICT (content): Merge conflict in App.java
Automatic merge failed; fix conflicts and then commit the result.
```

Error ini menunjukkan bahwa terdapat konflik dalam file ‘App.java‘ yang perlu diselesaikan sebelum melakukan push kembali.

7.11.6 Menampilkan File yang Konflik di Git

Ketika terjadi konflik selama proses `merge` atau `rebase`, Git akan menandai file yang mengalami konflik. Berikut adalah langkah-langkah untuk menampilkan file yang konflik:

1. Jalankan perintah berikut untuk menampilkan status repositori:

```
    git status
```

   Perintah ini akan menampilkan status direktori kerja dan area staging, termasuk file-file yang mengalami konflik. Cari baris yang menunjukkan ”both modified” atau ”unmerged” files.

2. Untuk menampilkan hanya nama file yang mengalami konflik, gunakan perintah berikut:

```
    git diff --name-only --diff-filter=U
```

   Perintah ini akan menampilkan daftar nama file yang memiliki konflik (ditandai sebagai ”U” untuk unmerged).

3. Jika ingin melihat perbedaan spesifik dalam file-file yang konflik, jalankan:

```
    git diff
```

   Perintah ini akan menunjukkan perubahan dari kedua cabang yang menyebabkan konflik, menggunakan penanda konflik (`<<<<<<<`, `=======`, `>>>>>>>`) dalam output.

7.11.7 Menyelesaikan Konflik

Buka file ‘App.java‘ dan perbaiki konflik yang ada. Kode target yang benar setelah perbaikan adalah sebagai berikut:

```
package com.example;

/**
* Hello world!
*
*/
public class App
{
  public static void main( String[] args )
  {
    System.out.println("BBBBBBBBb Hello World!");
    System.out.println( "AAAAAAAAAAa Hello World!" );
  }
}
```

Kemudian, lakukan langkah-langkah berikut untuk menyelesaikan proses:

1. Tambahkan file yang sudah diperbaiki ke staging area:

```
    git add App.java
```

2. Lakukan commit untuk menyelesaikan merge:

```
git commit -m "Resolve merge conflict in App.java"
```

3. Lakukan push ke repository GitHub:

```
git push origin main
```

Dengan mengikuti langkah-langkah ini, pengguna dapat mengupdate kode, menangani error saat melakukan push dan pull, serta menyelesaikan konflik yang terjadi.

Bab 8

Latihan Branch and Pull Request

8.1 Membuat Branch Baru untuk Proyek Maven dan Melakukan `Push`

Saat bekerja dalam tim atau mengembangkan fitur baru, penting untuk membuat *branch* terpisah agar perubahan dapat dikelola dengan lebih baik tanpa mengganggu `main` branch. Berikut adalah langkah-langkah untuk membuat *branch* baru di Git menggunakan proyek Maven yang telah dibuat:

1. Pastikan bahwa Anda berada di direktori proyek Maven:

```
cd /path/to/your/maven-project
```

2. Buat *branch* baru menggunakan perintah `git branch` atau `git checkout -b` untuk langsung beralih ke branch tersebut:

```
git checkout -b nama_branch_baru
```

   Perintah ini akan membuat *branch* baru dengan nama `nama_branch_baru` dan langsung memindahkan Anda ke branch tersebut.

3. Setelah *branch* dibuat, Anda dapat mulai melakukan perubahan pada proyek Maven. Setiap perubahan yang Anda lakukan pada branch ini akan terisolasi dari `main` branch.

4. Untuk melihat daftar *branch* yang ada, gunakan perintah:

```
git branch
```

   Perintah ini akan menampilkan semua *branch* yang ada di repository Git, dengan *branch* aktif ditandai dengan simbol `*`.

8.1.1 Update Kode Proyek Maven

Setelah membuat branch baru dan melakukan beberapa perubahan, berikut adalah update terbaru dari kode Java di proyek Maven. Kode ini sekarang memeriksa apakah ada argumen yang diberikan saat aplikasi dijalankan. Jika ada argumen, aplikasi akan menyapa pengguna dengan nama yang diberikan; jika tidak ada, aplikasi akan menampilkan pesan ”Hello, World!”.

```
package com.example;

public class App {
  public static void main(String[] args) {
    if (args.length > 0) {
      System.out.println("Hello, " + args[0] + "!");
    } else {
```

```
        System.out.println("Hello, World!");
      }
    }
  }
```

Perubahan ini menambahkan logika sederhana untuk menyesuaikan output berdasarkan argumen yang diberikan.

Setelah kode selesai, Anda dapat membangun proyek dengan menjalankan perintah:

```
mvn package
```

Maven akan mengompilasi proyek dan menghasilkan file `JAR` di direktori `target`.

Jalankan proyek dengan perintah berikut:

```
java -cp target/hello-world-1.0-SNAPSHOT.jar com.example.App John
```

Output programnya akan menjadi:

```
Hello, John!
```

Jika aplikasi dijalankan tanpa argumen:

```
Hello, World!
```

8.1.2 Menambahkan Perubahan, `Commit`, dan Mendorong (*Push*) Branch Baru ke GitHub

Setelah melakukan perubahan di branch baru, Anda dapat menambahkannya ke `staging area`, melakukan `commit`, dan kemudian mendorong (*push*) branch tersebut ke GitHub.

1. Tambahkan perubahan ke `staging area`:

   ```
       git add .
   ```

2. Lakukan `commit` dengan pesan yang sesuai:

   ```
       git commit -m "Menambahkan fitur baru pada Maven project"
   ```

3. Setelah `commit`, dorong (*push*) branch baru ke GitHub dengan perintah:

   ```
       git push origin nama_branch_baru
   ```

   Perintah ini akan mendorong *branch* baru yang telah dibuat dan di-`commit` ke remote repository di GitHub.

4. Jika perubahan sudah siap untuk digabungkan kembali ke `main`, branch ini dapat di-`merge` menggunakan perintah berikut (lakukan ini setelah beralih kembali ke `main` branch):

   ```
       git checkout main
       git merge nama_branch_baru
   ```

5. Setelah digabung, jangan lupa untuk mendorong perubahan dari `main` ke GitHub:

   ```
       git push origin main
   ```

Dengan menggunakan branch terpisah, Anda dapat mengembangkan fitur baru atau melakukan perbaikan tanpa mengganggu stabilitas proyek di branch utama, serta mendorong perubahan tersebut ke GitHub untuk dibagikan atau ditinjau oleh tim.

8.2 Membuat Pull Request di GitHub

Setelah melakukan perubahan pada branch baru dan melakukan push perubahan ke GitHub, langkah berikutnya adalah membuat *pull request* (PR) agar perubahan dapat ditinjau dan digabungkan ke cabang utama (main). Pull request dapat dibuat baik melalui GitHub web interface maupun command prompt menggunakan GitHub CLI (gh).

8.2.1 Membuat Pull Request Menggunakan GitHub Web

1. Buka repository proyek di GitHub yang telah Anda push ke branch baru.
2. Klik pada tab `Pull requests`.
3. Klik tombol `New pull request` di sisi kanan.
4. Pilih branch yang baru saja Anda push sebagai sumber (*source branch*), dan pastikan `main` sebagai tujuan (*target branch*).
5. Berikan deskripsi yang jelas tentang perubahan yang Anda buat, seperti:

   > "Mengubah `App.java` untuk mendukung argumen pengguna dan menampilkan pesan `Hello, <nama>!` jika argumen diberikan."

6. Klik tombol `Create pull request` untuk mengirimkan permintaan.

8.2.2 Membuat Pull Request Menggunakan Command Prompt

Anda juga dapat membuat pull request dari command prompt menggunakan GitHub CLI. Berikut adalah langkah-langkahnya:

1. Pastikan Anda telah menginstal GitHub CLI (gh) dan login ke GitHub menggunakan perintah berikut:

```
gh auth login
```

2. Setelah melakukan push ke branch baru, buat pull request menggunakan perintah:

```
gh pr create --base main --head <nama-branch> --title "Deskripsi PR"
    --body "Deskripsi detail tentang perubahan"
```

   Misalnya:

```
gh pr create --base main --head feature-branch --title "Update App.
    java" --body "Menambahkan dukungan untuk argumen pengguna pada App
    .java"
```

3. Perintah ini akan membuat pull request dari branch `feature-branch` ke branch `main`.
4. Jika berhasil, GitHub CLI akan memberikan URL yang dapat Anda buka untuk meninjau dan mengelola pull request tersebut.

8.2.3 Review dan Merge Pull Request

Setelah pull request dibuat, tim atau kontributor lain dapat meninjau perubahan yang telah Anda buat. Berikut adalah langkah-langkah lanjutan:

- Jika tidak ada konflik dan perubahan sudah sesuai, pull request dapat disetujui (*approved*).
- Setelah disetujui, pull request dapat digabungkan (*merged*) ke cabang utama (`main`) dengan mengklik tombol `Merge pull request` di GitHub, atau menggunakan GitHub CLI:

```
gh pr merge <nomor-PR>
```

- Jika ada konflik atau umpan balik dari peninjau, perbaiki perubahan di branch yang sama, kemudian push ulang ke branch tersebut untuk memperbarui pull request.

8.2.4 Menutup Pull Request

Setelah pull request digabungkan, Anda dapat menghapus branch yang sudah tidak dibutuhkan dengan mengklik tombol `Delete branch` di GitHub, atau dengan menjalankan perintah berikut di lokal:

```
1 git branch -d <nama-branch>
```

Ini memastikan bahwa repository tetap bersih dan terorganisir dengan baik.