GitCloneService

September 5, 2026

GitCloneService

Kind: Service

Source: `atloria-monorepo/apps/api/src/parsing/git-clone.service.ts`

`GitCloneService` is a NestJS backend service responsible for cloning remote Git repositories into local working directories for parsing and analysis. It validates repository access, builds authenticated clone URLs, discovers branches, retrieves the cloned repository's HEAD SHA, and removes temporary or stale clone directories.

Methods

Method	Signature	Returns	Description
`clone`	`clone(options: CloneOptions)`	`Promise<CloneResult>`	Clone a git repository
`testAccess`	`testAccess(options: {		

```
repositoryUrl: string;
provider: RepositoryProvider;
credentials?: RepositoryCredentialsDto;
```

}) | Promise<{ success: boolean; error?: string }> | Test repository access using git ls-remote (lightweight, no clone) | | listBranchesViaGit | listBranchesViaGit(options: {
repositoryUrl: string;
provider: RepositoryProvider;
credentials?: RepositoryCredentialsDto;
}) | Promise<{ success: boolean; branches: { name: string }[]; error?: string }> | List branches via git ls-remote (works for public repos without OAuth) |
| buildCloneUrl | buildCloneUrl(options: {
repositoryUrl: string;
provider: RepositoryProvider;

credentials?: RepositoryCredentialsDto;
}) | string | `Build clone URL with credentials embedded` |
parseRepositoryUrl	parseRepositoryUrl(repositoryUrl: string)	ParsedRepoUrl	`Parse repository URL to extract metadata`		getHeadSha	getHeadSha(localPath: string)	Promise<string	null>	`The HEAD commit SHA of a cloned working tree — the baseline docs are generated from, used later to compute an authoritative code diff on push.`
cleanup	cleanup(localPath: string)	Promise	`Cleanup cloned repository`
cleanupOldClones	cleanupOldClones(olderThanHours: number)	Promise`	Cleanup all temporary clone directories older than specified hours

Where it refuses work

- `GitCloneService` stops the work with `Error` when `typeof branch !== 'string' || !/^[A-Za-z0-9][\w./+@-]{0,254}$/.test(branch)`.
- `GitCloneService` stops the work with `Error` when `u.protocol !== 'https:' && u.protocol !== 'http:'` — “Only http(s) repository URLs are supported”.
- `GitCloneService` stops the work with `Error` when `provider === RepositoryProvider.AZURE_DEVOPS && !credentials` — “Azure DevOps repositories require authentication credentials”.
- `GitCloneService` stops the work with an early return when `!credentials || (!credentials.username && !credentials.password)`.

When something fails

- `GitCloneService` handles failure in 12 places: it turns it into a return value in 5, lets it reach the caller in 4, and logs it and continues in 3.

Diagram

```
sequenceDiagram
    participant Caller as Parsing Workflow
    participant Service as GitCloneService
    participant Git as Git CLI / Remote Provider
    participant FS as Local Filesystem

    Caller->>Service: testAccess()
    Service->>Git: Validate remote repository access
    Git-->>Service: success or error
    Service-->>Caller: { success, error? }

    Caller->>Service: clone()
    Service->>Service: parseRepositoryUrl()
```

```
    Service->>Service: buildCloneUrl()
    Service->>Git: git clone <authenticated-url>
    Git->>FS: Create local clone directory
    Git-->>Service: CloneResult
    Service-->>Caller: clone result

    Caller->>Service: listBranchesViaGit()
    Service->>Git: Query remote/local branches
    Git-->>Service: Branch names
    Service-->>Caller: { success, branches }

    Caller->>Service: getHeadSha()
    Service->>Git: git rev-parse HEAD
    Git-->>Service: SHA or null
    Service-->>Caller: HEAD SHA

    Caller->>Service: cleanup()
    Service->>FS: Remove clone directory
```

Usage

```typescript
import { Injectable } from '@nestjs/common';
import { GitCloneService } from './git-clone.service';

@Injectable()
export class RepositoryParsingService {
  constructor(private readonly gitCloneService: GitCloneService) {}

  async parseRepository() {
    const access = await this.gitCloneService.testAccess();

    if (!access.success) {
      throw new Error(`Unable to access repository: ${access.error}`);
    }

    try {
      const cloneResult = await this.gitCloneService.clone();
      const headSha = await this.gitCloneService.getHeadSha();
      const branches = await this.gitCloneService.listBranchesViaGit();

      return {
        clonePath: cloneResult.path,
        headSha,
        branches: branches.branches.map((branch) => branch.name),
      };
    } finally {
      await this.gitCloneService.cleanup();
    }
```

```
  }
}
```

AI Coding Instructions

- Call `testAccess()` before cloning when repository credentials or permissions may be invalid, and surface its error message to the caller.
- Always wrap clone-dependent work in `try/finally` and call `cleanup()` to avoid leaving temporary repositories on disk.
- Use `buildCloneUrl()` and `parseRepositoryUrl()` rather than manually constructing provider-specific Git URLs or parsing repository identifiers.
- Treat branch listing and HEAD SHA retrieval as Git operations that can fail independently after a successful clone.
- Use `cleanupOldClones()` in scheduled maintenance or startup workflows to remove stale clone directories.

Referenced By

- `DocsPrService` (DEPENDS_ON)
- `ParsingModule` (MODULE_PROVIDES)
- `ParsingModule` (MODULE_EXPORTS)
- `TestConnectionController` (DEPENDS_ON)
- `TechnicalDocsParserService` (DEPENDS_ON)