AI VOID FIELD GUIDE

Cut the Chase

No-nonsense, code-first guides that get straight to the point. Zero fluff, maximum value.

AI VOID

2026-05-04 18:30 IST

Contents

CHAPTER 01

Component Craft: Angular UI Library Forge - Fast-Track UI Libraries

Angular v18.x, Angular CLI v18.x (as of 2026-02-10)

Workspace & Library Generation

Begin by creating a new Angular monorepo workspace without an initial application, then generate your UI library.

```
ng new my-ui-workspace --no-create-application --strict --standalone false
--routing false --style scss
// Creates a new Angular workspace without a root application.
// --no-create-application: Skips initial app creation.
// --strict: Enables strict type checking.
// --standalone false: Uses NgModules (still common for libraries, though
standalone is default for apps).
// --routing false: No routing module.
// --style scss: Preferred stylesheet format.

cd my-ui-workspace
// Navigate into the new workspace directory.

ng generate library my-ui-components --prefix mui --skip-install
// Generates a new library project named 'my-ui-components'.
// --prefix mui: Sets the prefix for generated components (e.g., <mui-button>).
// --skip-install: Skips running npm install after generation (useful for
monorepos).
```

Component Structure & Public API

Organize your library components and expose them through the `public-api.ts` file, which defines the library's public interface.

```
// projects/my-ui-components/src/lib/button/button.component.ts
import { Component, Input, Output, EventEmitter, HostBinding } from '@angular/
core';

@Component({
  selector: 'mui-button', // Component selector, using library prefix
  template: `
    <button [type]="type" [disabled]="disabled" (click)="onClick.emit($event)">
      <ng-content></ng-content> <!-- Allows projected content inside the button
-->
    </button>
  `,
  styleUrls: ['./button.component.scss'],
  standalone: true // Libraries increasingly use standalone components for
better tree-shaking
})
export class ButtonComponent {
  @Input() type: 'button' | 'submit' | 'reset' = 'button'; // Input for button
type
  @Input() disabled = false; // Input to disable the button
  @Output() onClick = new EventEmitter<Event>(); // Output for click events

  @HostBinding('class.mui-button-disabled') // Adds a class when disabled
  get isDisabledClass(): boolean {
    return this.disabled;
  }
}

// projects/my-ui-components/src/public-api.ts
/*
 * Public API Surface of my-ui-components
 */

export * from './lib/button/button.component'; // Export the standalone
component
// For NgModule-based libraries, you would export the module:
// export * from './lib/my-ui-components.module';
```

Inputs, Outputs & Host Bindings

Components interact via `@Input` for data flow into the component, `@Output` for events flowing out, and `@HostBinding` to dynamically apply host element properties.

```
// projects/my-ui-components/src/lib/card/card.component.ts
import { Component, Input, HostBinding } from '@angular/core';

@Component({
  selector: 'mui-card',
  template: `
    <div class="mui-card-header" *ngIf="header">
      <h3>{{ header }}</h3>
    </div>
    <div class="mui-card-content">
      <ng-content></ng-content> <!-- Content projection for card body -->
    </div>
  `,
  styleUrls: ['./card.component.scss'],
  standalone: true
})
export class CardComponent {
  @Input() header: string | null = null; // Optional header text
  @Input() elevation: 1 | 2 | 3 | 4 = 1; // Input for card shadow level

  @HostBinding('class.mui-card') readonly hostClass = true; // Always add base
class
  @HostBinding('attr.role') readonly hostRole = 'region'; // Accessibility role

  @HostBinding('class.mui-card-elevation-1') get isElevation1() { return this.e
levation === 1; }
  @HostBinding('class.mui-card-elevation-2') get isElevation2() { return this.e
levation === 2; }
  @HostBinding('class.mui-card-elevation-3') get isElevation3() { return this.e
levation === 3; }
  @HostBinding('class.mui-card-elevation-4') get isElevation4() { return this.e
levation === 4; }
}

// projects/my-ui-components/src/public-api.ts (updated)
export * from './lib/button/button.component';
export * from './lib/card/card.component'; // Expose the new component
```

Theming & Accessibility

Implement theming using CSS custom properties for flexibility and ensure basic accessibility attributes are present.

```scss
// projects/my-ui-components/src/lib/button/button.component.scss
:host {
  display: inline-block; // Ensure host element behaves correctly
}

button {
  --mui-button-bg: var(--mui-primary-color, #007bff); // Default background, ca
n be overridden
  --mui-button-text-color: var(--mui-text-color-light, #ffffff); // Default tex
t color
  --mui-button-border-radius: 4px;

  padding: 8px 16px;
  border: none;
  border-radius: var(--mui-button-border-radius);
  background-color: var(--mui-button-bg);
  color: var(--mui-button-text-color);
  cursor: pointer;
  font-size: 1rem;
  transition: background-color 0.2s ease;

  &:hover {
    filter: brightness(0.9);
  }

  &[disabled] {
    opacity: 0.6;
    cursor: not-allowed;
  }
}

// projects/my-ui-components/src/lib/card/card.component.ts (Accessibility
update)
import { Component, Input, HostBinding } from '@angular/core';

@Component({
  selector: 'mui-card',
  template: `
    <div class="mui-card-header" *ngIf="header">
      <h3 [attr.id]="headerId">{{ header }}</h3> <!-- Link header to card conte
nt -->
    </div>
    <div class="mui-card-content" [attr.aria-labelledby]="header ? headerId :
null">
      <ng-content></ng-content>
    </div>
  `,
  styleUrls: ['./card.component.scss'],
  standalone: true
})
export class CardComponent {
  @Input() header: string | null = null;
  @Input() elevation: 1 | 2 | 3 | 4 = 1;

  @HostBinding('class.mui-card') readonly hostClass = true;
  @HostBinding('attr.role') readonly hostRole = 'region';

  readonly headerId = `mui-card-header-$
{Math.random().toString(36).substring(2, 9)}`; // Unique ID for accessibility
```

```
  // ... (elevation HostBindings remain the same)
}
```

Packaging & Distribution (Monorepo)

Build the library for internal consumption within the same workspace. The Angular CLI automatically configures `tsconfig.json` paths.

```
# Build the library for production
ng build my-ui-components --configuration production
# This command compiles the library, outputs to dist/my-ui-components.
# The --configuration production flag applies production optimizations.

# Verify the output structure
ls -l dist/my-ui-components
# Expected: bundles (fesm2022, esm2022, umd), package.json, typings (.d.ts).
```

Packaging & Distribution (NPM)

Prepare the library for publishing to a public or private npm registry, correctly managing `peerDependencies`.

```
// projects/my-ui-components/package.json
{
  "name": "@my-scope/my-ui-components", // Scoped package name for npm
  "version": "0.1.0", // Semantic versioning (major.minor.patch)
  "peerDependencies": { // Critical: Dependencies that the CONSUMING app must
provide
    "@angular/common": "^18.0.0", // Match Angular version of the library
    "@angular/core": "^18.0.0"
  },
  "dependencies": { // Only include truly internal, non-Angular dependencies
    "tslib": "^2.3.0" // Angular libraries typically depend on tslib
  },
  "sideEffects": false // Important for tree-shaking (all code is pure)
}
```

```
# Publish to npm (after building the library)
cd dist/my-ui-components
# Navigate to the built library's output directory.

npm publish --access public
# Publishes the package to the npm registry.
# --access public: Required for public packages. Omit for private packages.
# Ensure you are logged in to npm (npm login) or have configured a private
registry.
```

Dependency Management & Tree-shaking

Properly define `dependencies`, `devDependencies`, and `peerDependencies` in `package.json`. Use `sideEffects: false` to enable effective tree-shaking.

```
// projects/my-ui-components/package.json (revisited)
{
  "name": "@my-scope/my-ui-components",
  "version": "0.1.0",
  "peerDependencies": {
    "@angular/common": "^18.0.0", // Consuming app must provide Angular common
module
    "@angular/core": "^18.0.0"   // Consuming app must provide Angular core
module
  },
  "dependencies": {
    "tslib": "^2.3.0" // Only actual runtime dependencies *not* provided by
Angular itself
  },
  "devDependencies": {
    "@angular/compiler": "^18.0.0", // Used only during compilation, not
runtime
    "@angular/compiler-cli": "^18.0.0",
    "ng-packagr": "^18.0.0", // Tool for building Angular libraries
    "typescript": "~5.4.0"
  },
  "sideEffects": false, // Crucial for tree-shaking. Tells bundlers that this
package has no side effects.
                        // Allows removal of unused exports during bundling.
  "repository": {
    "type": "git",
    "url": "https://github.com/my-org/my-ui-components.git"
  },
  "keywords": ["angular", "ui", "components"],
  "license": "MIT"
}
```

Versioning & Build Order

Adopt Semantic Versioning (SemVer) for your library releases. In a monorepo, define scripts to manage build order if libraries depend on each other.

```
# Update library version (SemVer)
cd projects/my-ui-components
npm version patch --no-git-tag-version
# Increments the patch version (e.g., 0.1.0 -> 0.1.1).
# --no-git-tag-version: Prevents npm from creating a git tag, useful in CI/CD.
# Other options: minor, major.

# Example package.json scripts for monorepo build order
// package.json (root workspace)
{
  "name": "my-ui-workspace",
  "version": "0.0.0",
  "scripts": {
    "build:my-ui-components": "ng build my-ui-components --configuration
production",
    "build:my-app": "ng build my-app --configuration production",
    "build:all": "npm run build:my-ui-components && npm run build:my-app",
    "test:my-ui-components": "ng test my-ui-components",
    "test:my-app": "ng test my-app"
  }
}
# Execute: npm run build:all
# This ensures the library is built before the application that consumes it.
```

Consuming the Library (Various Scenarios)

Integrate your UI library into Angular applications using different methods based on your project setup.

1. Monorepo Consumption (Recommended)

The Angular CLI automatically configures `tsconfig.json` paths for libraries within the same workspace.

```
// apps/my-app/src/app/app.module.ts (or app.component.ts if standalone)
import { Component } from '@angular/core';
import { ButtonComponent } from '@my-scope/my-ui-components'; // Import
directly from library name
import { CardComponent } from '@my-scope/my-ui-components';

@Component({
  selector: 'app-root',
  template: `
    <h1>My App</h1>
    <mui-button (onClick)="showAlert()">Click Me</mui-button>
    <mui-card header="Welcome Card" [elevation]="2">
      <p>This is content inside the card.</p>
    </mui-card>
  `,
  standalone: true,
  imports: [ButtonComponent, CardComponent] // Import standalone components
})
export class AppComponent {
  showAlert() {
    alert('Button clicked!');
  }
}

// tsconfig.json (root workspace) - CLI handles this automatically
{
  "compilerOptions": {
    "paths": {
      "@my-scope/my-ui-components": [
        "dist/my-ui-components"
      ],
      "@my-scope/my-ui-components/*": [
        "dist/my-ui-components/*"
      ]
    }
  }
}
```

2. Local Linking (Development)

For developing an application that consumes a library not in the same workspace, `npm link` creates symlinks.

```
# In the library's dist directory (after ng build)
cd dist/my-ui-components
npm link
# Creates a global symlink for your library.

# In the consuming application's root directory
cd ../../apps/my-other-app # Example: outside the monorepo
npm link @my-scope/my-ui-components
# Links the global symlink to this application's node_modules.

# To unlink later
cd apps/my-other-app
npm unlink @my-scope/my-ui-components
cd dist/my-ui-components
npm unlink
```

3. Tarball Install (Testing/Private Distribution)

Install a `.tgz` package directly, useful for testing specific builds or private distribution without a registry.

```
# In the library's dist directory (after ng build)
cd dist/my-ui-components
npm pack
# Creates a tarball, e.g., my-scope-my-ui-components-0.1.0.tgz

# In the consuming application's root directory
cd ../../apps/my-other-app
npm install ../../dist/my-ui-components/my-scope-my-ui-components-0.1.0.tgz
# Installs the tarball. Update path and version as needed.
```

4. Private NPM Registry

Install from a private registry after publishing.

```
# In the consuming application's root directory
npm install @my-scope/my-ui-components@0.1.0
# Ensure your npm client is configured to access the private registry.
# (e.g., via .npmrc file or npm login)
```

CI/CD Integration & Optimization

Automate library builds, tests, and integrate into application deployments. Optimize for production.

```
// package.json (root workspace) - Example CI/CD scripts
{
  "name": "my-ui-workspace",
  "version": "0.0.0",
  "scripts": {
    "build:lib": "ng build my-ui-components --configuration production",
    "test:lib": "ng test my-ui-components --watch=false --
browsers=ChromeHeadless",
    "lint:lib": "ng lint my-ui-components",
    "publish:lib": "cd dist/my-ui-components && npm publish --access public", /
/ Or private
    "build:app": "ng build my-app --configuration production",
    "test:app": "ng test my-app --watch=false --browsers=ChromeHeadless",
    "lint:app": "ng lint my-app",
    "ci:build-and-test": "npm run build:lib && npm run test:lib && npm run
build:app && npm run test:app",
    "ci:deploy": "npm run build:app && firebase deploy" // Example deployment
  }
}
```

```
// angular.json (for application build optimization)
{
  "projects": {
    "my-app": {
      "architect": {
        "build": {
          "options": {
            "optimization": true, // Enables various optimizations
(minification, tree-shaking)
            "outputHashing": "all", // Cache busting for deployed files
            "sourceMap": false, // Disable for production builds
            "namedChunks": false, // Smaller bundles, less readable names
            "extractLicenses": true, // Extract third-party licenses
            "vendorChunk": false, // Consolidate vendor code into main bundle
            "buildOptimizer": true, // Advanced optimizations for smaller
bundles
            "budgets": [ // Define performance budgets to prevent regressions
              {
                "type": "initial",
                "maximumWarning": "500kb",
                "maximumError": "1mb"
              },
              {
                "type": "anyComponentStyle",
                "maximumWarning": "2kb",
                "maximumError": "4kb"
              }
            ]
          },
          "configurations": {
            "production": {
              "aot": true, // Always enable Ahead-of-Time compilation for
production
              "serviceWorker": false, // Enable if PWA is desired
              "fileReplacements": [
                {
                  "replace": "apps/my-app/src/environments/environment.ts",
                  "with": "apps/my-app/src/environments/environment.prod.ts"
                }
              ]
            }
          }
        }
      }
    }
  }
}
```

Quick Reference

- `ng generate library <name>`: Creates a new library project.
- `public-api.ts`: Defines the public interface of your library. All exports from here are part of the consumable API.
- `@Input()` / `@Output()`: Decorators for component communication.

- `@HostBinding()` / `@HostListener()`: Interact with the host element.
- `peerDependencies`: Crucial in `package.json` for libraries; specifies dependencies the consumer must provide.
- `sideEffects: false`: In `package.json`, enables bundlers to perform aggressive tree-shaking.
- `ng build <library-name> --configuration production`: Builds the library with production optimizations.
- `npm link`: For local development linking of libraries.
- `npm publish`: Publishes library to npm registry.
- `optimization: true`, `aot: true`: Key `angular.json` build options for production performance.
- **Angular v18.x**: Current stable version as of 2026-02-10, with increased adoption of standalone components and signals.

References

1. Angular Libraries Guide
2. Angular CLI Workspace and Project File Structure
3. Angular CLI `generate library` Command
4. Angular Best Practices
5. npm `package.json` `peerDependencies`

This page is AI-assisted. References official documentation.

CHAPTER 02

AWK Demystified - Text Processing Essentials

GNU Awk (gawk) 5.3.0 (stable as of late 2025) is the primary implementation.

Core Syntax

AWK processes input line by line, executing `action` blocks when `pattern` matches. `BEGIN` and `END` blocks run before and after file processing, respectively.

```
# Basic structure: 'pattern { action }'
# Prints every line (default action if none specified)
awk '{ print }' data.txt

# Prints lines containing "error"
awk '/error/ { print }' log.txt

# BEGIN block: executed once before any input is read
# END block: executed once after all input is processed
awk 'BEGIN { print "--- Log Analysis Start ---" } /FAIL/ { count++ } END
{ print "Total failures:", count }' system.log
```

Field Handling & Built-in Variables

AWK automatically splits each input line into fields. `$0` is the entire line, `$1` is the first field, `$2` the second, and so on. `NF` is the number of fields, `NR` is the current record (line) number, `FS` is the field separator (default space/tab), `OFS` is the output field separator (default space).

```
# Print the second and first fields, separated by a comma
awk '{ print $2, $1 }' names.txt

# Print the last field of each line
awk '{ print $NF }' data.csv

# Change field separator to comma using -F option
awk -F',' '{ print $1, $3 }' data.csv

# Change field separator within the script (BEGIN block is ideal)
awk 'BEGIN { FS=":"; OFS=" -> " } { print $1, $3 }' /etc/passwd

# Print line number and the entire line
awk '{ print NR, $0 }' file.txt

# Process only lines with at least 3 fields
awk 'NF >= 3 { print $1, $NF }' data.log
```

Conditional Logic & Loops

AWK supports `if/else`, `for`, and `while` loops for flow control. `next` skips to the next input record, `exit` terminates processing.

```
# If-else statement: check a condition on a field
awk '{
    if ($3 > 100) {
        print "High value:", $0
    } else {
        print "Normal value:", $0
    }
}' metrics.txt

# For loop: iterate through fields
awk '{
    for (i = 1; i <= NF; i++) { # Loop from first field to last
        print "Field", i, ":", $i
    }
}' single_line.txt

# While loop: example for specific conditions (less common than for)
awk '{
    i = 1
    while (i <= NF && length($i) < 5) { # Process fields while length is less
than 5
        print "Short field:", $i
        i++
    }
}' words.txt

# 'next' statement: skip remaining actions for current record
awk '/^#/ { next } { print $0 }' config.ini # Skip comment lines

# 'exit' statement: stop processing immediately
awk '/ERROR/ { print "Found error, exiting."; exit } { print $0 }' log.txt
```

Functions & Arrays

AWK supports user-defined functions and powerful associative arrays (hash maps).

```
# User-defined function: calculate average
awk '
function average(a, b) { # Define a function 'average'
    return (a + b) / 2
}
{
    avg = average($1, $2) # Call the function
    print $1, $2, avg
}' numbers.txt

# Associative arrays: count occurrences of field values
awk '{
    count[$1]++ # Increment count for the value of the first field
}
END {
    for (item in count) { # Iterate through array keys
        print item, count[item]
    }
}' access.log

# Associative arrays for summing values by key
awk '{
    sum[$1] += $2 # Add second field to sum for key (first field)
}
END {
    for (key in sum) {
        print key, sum[key]
    }
}' sales.csv
```

Output Formatting

The `printf` function provides C-style formatted output, offering more control than `print`.

```
# printf for formatted output
awk '{
    printf "Item: %-10s Price: $%5.2f Quantity: %d\n", $1, $2, $3
}' inventory.txt

# Example with padding and precision
# %-10s: left-justified string, 10 chars wide
# %5.2f: float, 5 total width (including decimal), 2 decimal places
# %d: integer
```

Real-world Log Processing

AWK excels at parsing structured log files.

```
# Example: Extract IP, timestamp, and request from Apache access logs
# Log format: 192.168.1.1 - - [29/Dec/2025:10:00:00 +0000] "GET /index.html
HTTP/1.1" 200 1234
awk -F'[][]' '{ # Split by square brackets
    ip = $1; sub(/ - - $/, "", ip) # Clean up IP field
    timestamp = $2 # Timestamp is in second field
    request = $3; sub(/^"|"$ /, "", request) # Clean up request field
    print "IP:", ip, "Time:", timestamp, "Request:", request
}' access.log

# Example: Sum bytes transferred by IP
awk '{
    ip = $1
    bytes = $NF # Last field is bytes transferred
    if (bytes ~ /^[0-9]+$/) { # Ensure bytes is a number
        total_bytes[ip] += bytes
    }
}
END {
    print "--- Bytes Transferred by IP ---"
    for (addr in total_bytes) {
        printf "%-15s %10d bytes\n", addr, total_bytes[addr]
    }
}' access.log
```

Performance Considerations

For very large files, performance matters.

```
# Pre-filter with grep for massive files to reduce AWK's workload
# This is often faster than AWK's pattern matching on its own for simple regex
grep "specific_pattern" large_file.log | awk '{ print $1, $NF }'

# Use 'next' early to avoid unnecessary processing
# If a line doesn't meet initial criteria, skip it immediately
awk '{
    if ($1 == "SKIP") { next } # Skip lines starting with "SKIP"
    # ... rest of complex processing ...
    print $0
}' data.txt

# Avoid unnecessary regex matching in loops
# Pre-compile regex if possible (gawk specific, `~` operator is fast enough for
most)
# For very complex patterns, consider using `match()` and `RSTART`/`RLENGTH`
```

Gotchas & Best Practices

- **`FS` vs `-F`**: Setting `FS` in `BEGIN` block is generally cleaner and more robust than `-F` for complex separators.
- **Default Action**: If no action is specified for a pattern, `print $0` is the default.
- **Quoting**: Shell expansion can interfere. Always single-quote AWK scripts to prevent shell interpretation of `$`, `*`, etc.
- **Variable Scope**: All variables are global by default. Use parameter lists in user-defined functions to declare local variables (e.g., `function my_func(arg, local_var1, local_var2)`).
- **Empty Fields**: When `FS` is a single character, consecutive separators result in empty fields (e.g., `a,,b` with `FS=","` gives `$2=""`). If `FS` is a regex, multiple separators are treated as one.
- **Performance with `printf`**: `printf` can be slightly slower than `print` due to formatting overhead, but usually negligible.

Quick Reference

- **Command:** `awk 'script' file(s)` or `awk -f script_file file(s)`
- **Built-in Variables:**
  - `$0`: Entire line
  - `$1, $2, ...`: Fields
  - `NF`: Number of fields
  - `NR`: Record (line) number
  - `FS`: Input Field Separator (default: whitespace)
  - `OFS`: Output Field Separator (default: space)
  - `RS`: Record Separator (default: newline)
  - `ORS`: Output Record Separator (default: newline)
  - `FILENAME`: Current input filename
- **Patterns:** `/regex/`, `expression`, `BEGIN`, `END`, `pattern1, pattern2` (range)
- **Actions:** `{ statements }`
- **Control Flow:** `if/else`, `for`, `while`, `next`, `exit`

- **Functions:** `length()`, `sub()`, `gsub()`, `split()`, `substr()`, `index()`, `match()`, `sprintf()`, `system()`, `atan2()`, `cos()`, `sin()`, `exp()`, `log()`, `sqrt()`, `rand()`, `srand()`, `int()`, `toupper()`, `tolower()`
- **Operators:** Arithmetic (`+ - * / % ^`), Comparison (`== != < > <= >=`), Logical (`&& || !`), Assignment (`= += -= *= /= %= ^=`), Concatenation (space)
- **Version:** GNU Awk (gawk) 5.3.0 (as of 2025-12-29)

References

1. GNU Awk User's Guide - The definitive guide for gawk.
2. AWK - A Tutorial and Introduction - A comprehensive, classic tutorial.
3. The AWK Programming Language (Book) - The original book by Aho, Weinberger, Kernighan.

This page is AI-assisted. References official documentation.

CHAPTER 03

Find Command Mastery - Linux File Search Essentials

The `find` command is a powerful utility for locating files and directories in a filesystem hierarchy. (GNU findutils 4.9.0, as of late 2025)

Core Syntax

The fundamental structure of `find` involves a starting directory, followed by expressions that define search criteria and actions.

```
find . -name "myfile.txt" # Search current directory for a file named
"myfile.txt"
find /var/log -type f     # Find all regular files within /var/log
find /home -type d        # Find all directories within /home
```

Essential Patterns

Locating files based on common attributes like name (case-insensitive), modification time, or size are frequent operations.

```
# Find files with a specific name, ignoring case
find . -iname "report.pdf" # -iname performs a case-insensitive search

# Find files modified in the last 7 days
find /data -mtime -7       # -mtime N: files modified N*24 hours ago. -N for
less than N, +N for more than N.

# Find files larger than 100MB
find /tmp -size +100M      # +100M for >100MB, -100M for <100MB, 100M for
exactly 100MB. K, M, G suffixes.
```

Common Use Cases

Beyond simple searches, `find` excels at combining criteria and executing actions on the found items.

```
# Find all .log files in /var/log and delete them (use with extreme caution!)
find /var/log -name "*.log" -type f -delete # -delete is a GNU find extension,
efficient but irreversible.

# Find all empty directories in /tmp and remove them
find /tmp -type d -empty -delete # -empty finds empty files or directories.

# Find all Python files in the current directory and change their permissions
to 644
find . -name "*.py" -type f -exec chmod 644 {} \; # -exec runs a command for
each found item. {} is a placeholder for the filename, \; terminates the
command.

# Find files owned by user 'olduser' and change ownership to 'newuser'
find /srv/data -user olduser -exec chown newuser {} +
# Using + passes multiple arguments to -exec for efficiency.
```

Gotchas & Best Practices

Incorrect quoting or handling of special characters can lead to unexpected results. Always consider security and performance.

```
# CORRECT: Quoting wildcards prevents shell expansion, letting find handle them
find . -name "*.txt" # Shell passes "*.txt" literally to find.

# INCORRECT: Shell expands "*.txt" before find sees it if matching files exist
# find . -name *.txt # If "a.txt" and "b.txt" exist, find sees "find . -name
a.txt b.txt"

# Handling filenames with spaces or special characters safely
# Use -print0 with xargs -0 to handle null-terminated strings
find /path/to/files -name "*.bak" -print0 | xargs -0 rm # Safely deletes files
even with spaces/newlines.

# Avoid searching root (/) unless necessary; specify a narrower path for
performance
find /var/www -name "index.html" # Good: focused search
# find / -name "index.html" # Bad: scans entire filesystem, slow.
```

Advanced Techniques

For complex scenarios, `find` offers sophisticated options for logical operations, permissions, and depth control.

```
# Find files not owned by 'root' AND not modified in the last 30 days
find /var/www \( ! -user root -a ! -mtime -30 \) -type f # \( \) for grouping,
-a for AND (default), ! for NOT.

# Find files with execute permission for 'other' (o+x)
find . -perm /o+x -type f # /mode checks if ANY of the bits in mode are set.
-perm mode for EXACT match.

# Find files larger than 1GB OR modified within the last day
find /mnt/backup \( -size +1G -o -mtime -1 \) -type f # -o for OR.

# Limit search depth: only current directory (depth 1)
find . -maxdepth 1 -type f -name "*.conf" # -maxdepth N: don't descend more
than N levels.
find . -mindepth 2 -type d -name "cache" # -mindepth N: start processing at N
levels down.
```

Quick Reference

- **`find [path] [expression]`**: Basic syntax.
- **`-name "pattern"`**: Search by filename (case-sensitive).
- **`-iname "pattern"`**: Search by filename (case-insensitive).
- **`-type [f|d|l]`**: Search by type (file, directory, symlink).
- **`-mtime [+/-]N`**: Modified time (N*24h ago). `-N` (less than N), `+N` (more than N).
- **`-size [+/-]NC`**: File size (C=b, c, w, k, M, G). `+N` (larger), `-N` (smaller).
- **`-user USER`**: Files owned by `USER`.
- **`-group GROUP`**: Files owned by `GROUP`.
- **`-perm MODE`**: Files with specific permissions (e.g., `644`, `/u+x`).
- **`-empty`**: Empty files or directories.
- **`-delete`**: Delete found files/directories (GNU extension, use with caution).
- **`-exec CMD {} \;`**: Execute `CMD` for each result.
- **`-exec CMD {} +`**: Execute `CMD` with multiple results (more efficient).
- **`-print0`**: Print results null-terminated (for `xargs -0`).
- **`-maxdepth N`**: Limit search depth.
- **`-mindepth N`**: Start search at depth N.
- **`\( expr1 -a expr2 \)`**: Group expressions with AND (default).
- **`expr1 -o expr2`**: OR operator.
- **`! expr`**: NOT operator.

References

1. GNU findutils Manual
2. Oracle: Guide to Linux Find Command Mastery
3. Baeldung: Guide to the Linux find Command
4. Red Hat: 10 ways to use the Linux find command

This page is AI-assisted. References official documentation.

CHAPTER 04

Git Worktree Unlocked - Parallel Development Essentials

Git Worktree enables multiple working directories, each connected to the same repository but checked out to a different branch, all sharing the core object database. Git 2.44.0+ (as of 2026-03-07).

Core Concept: The Multi-Directory Model

Traditionally, `git checkout` changes your entire working directory and index. Git Worktree breaks this by allowing multiple working directories, each with its own `HEAD`, index, and working tree, all stemming from a single, shared `.git/objects` store. This solves the problem of needing to stash or commit incomplete work when switching contexts.

To visualize the internal structure:

```
# Initialize a new Git repository
git init my_project
cd my_project

# Create a dummy file and commit to 'main'
echo "Initial content" > file.txt
git add file.txt
git commit -m "Initial commit"

# List the contents of the main .git directory
# Note the absence of a 'worktrees' directory initially
ls -F .git
# Expected output (truncated):
# HEAD
# config
# description
# hooks/
# info/
# objects/
# refs/
```

Essential Patterns: Basic Worktree Operations

Creating a new worktree allows you to check out a branch into a separate directory. This directory will contain a minimal `.git` file pointing back to the main repository's `.git` directory, specifically to its `worktrees/<name>` subdirectory.

Create a new worktree for an existing branch:

```
# From the main repository's root (my_project/)
# Create a new worktree named 'feature-a' in a sibling directory '../feature-a-
worktree'
# and check out the 'main' branch into it.
git worktree add ../feature-a-worktree main
# Expected output:
# Preparing working tree (checking out 'main')
# HEAD is now at 8e3c0f2 Initial commit

# Now, list the worktrees
git worktree list
# Expected output:
# /path/to/my_project           8e3c0f2 [main]
# /path/to/feature-a-worktree   8e3c0f2 [main]

# Inspect the new worktree's .git directory
ls -F ../feature-a-worktree/.git
# Expected output:
# HEAD
# commondir
# gitdir
# index
# logs/
# ORIG_HEAD
# worktree
#
# Note: commondir points to the main .git, gitdir points to .git/worktrees/
feature-a-worktree
```

Create a new worktree and a new branch simultaneously:

```
# From the main repository's root (my_project/)
# Create a new worktree for a new branch 'dev-branch'
git worktree add ../dev-worktree dev-branch
# Expected output:
# Preparing working tree (new branch 'dev-branch')
# HEAD is now at 8e3c0f2 Initial commit

# List all worktrees again
git worktree list
# Expected output:
# /path/to/my_project           8e3c0f2 [main]
# /path/to/feature-a-worktree   8e3c0f2 [main]
# /path/to/dev-worktree         8e3c0f2 [dev-branch]
```

Removing a worktree:

```
# From the main repository's root (my_project/)
# Remove the 'feature-a-worktree' worktree.
# The directory must be empty or Git will refuse to remove it.
# It's safer to remove the directory manually *after* removing the worktree
reference.
git worktree remove ../feature-a-worktree
# Expected output:
# removed worktree ../feature-a-worktree

# Manually remove the physical directory if it still exists
rm -rf ../feature-a-worktree

# List worktrees to confirm removal
git worktree list
# Expected output:
# /path/to/my_project           8e3c0f2 [main]
# /path/to/dev-worktree         8e3c0f2 [dev-branch]
```

Common Use Cases: Streamlining Workflows

Worktrees excel in scenarios requiring rapid context switching or parallel operations without disrupting existing work.

Working on multiple features simultaneously:

```
# In main_project (on 'main' branch)
# You are working on 'feature-X' but need to start 'feature-Y'
git branch feature-X # Create feature branch
git checkout feature-X # Switch to feature-X in main worktree

# Create a new worktree for feature-Y
git worktree add ../feature-Y-worktree feature-Y
# Now you can work on feature-X in 'main_project' and feature-Y in 'feature-Y-
worktree' concurrently.
```

Reviewing a Pull Request (PR) without stashing:

```
# In main_project (on 'main' branch or any feature branch)
# You receive a PR for 'pr-branch-123' from 'origin'
git fetch origin pr-branch-123:pr-branch-123 # Fetch the PR branch locally

# Create a dedicated worktree for reviewing this PR
git worktree add ../pr-review-123 pr-branch-123
# Now navigate to ../pr-review-123, build, test, and review the PR code without
touching your current work.
```

Running parallel builds or tests:

```
# In main_project (on 'dev' branch)
# You need to run integration tests on 'dev' while also testing a new 'release-candidate' branch.
git worktree add ../release-candidate-tests release-candidate
# In one terminal: cd main_project && make test-dev
# In another terminal: cd ../release-candidate-tests && make test-rc
```

Gotchas & Best Practices

Branch Locking: A branch can only be checked out in one worktree at a time. Trying to `git checkout` a branch that's active in another worktree will fail.

```
# In main_project (on 'main')
git worktree add ../another-main main
# Expected output:
# fatal: 'main' is already checked out at '/path/to/my_project'
```

Deleting Branches: You cannot delete a branch (`git branch -d`) if it's currently checked out in any worktree. You must remove the associated worktree first.

```
# Assume 'dev-branch' is checked out in '../dev-worktree'
git branch -d dev-branch
# Expected output:
# fatal: Cannot delete branch 'dev-branch' checked out at '/path/to/dev-worktree'

# Correct sequence:
git worktree remove ../dev-worktree # Or navigate into ../dev-worktree and run `git worktree remove .`
rm -rf ../dev-worktree
git branch -d dev-branch
# Expected output:
# Deleted branch dev-branch (was 8e3c0f2).
```

Stale Worktree References: If you manually delete a worktree directory without using `git worktree remove`, Git's internal references in `.git/worktrees` become stale.

```
# Assume '../dev-worktree' was deleted manually (e.g., rm -rf ../dev-worktree)
# The reference still exists in the main repo's .git/worktrees/
git worktree list
# Expected output:
# /path/to/my_project           8e3c0f2 [main]
# /path/to/dev-worktree         (unavailable) # Indicates a stale entry

# Clean up stale references:
git worktree prune
# Expected output:
# Pruning worktrees...
# Removed worktree /path/to/dev-worktree
git worktree list
# Expected output:
# /path/to/my_project           8e3c0f2 [main]
```

Directory Organization: Keep worktrees logically organized. A common pattern is to create a dedicated parent directory for all worktrees, e.g., `~/projects/my_project` (main repo) and `~/projects/my_project_worktrees/feature-a`, `~/projects/my_project_worktrees/pr-review`.

Naming Conventions: Use descriptive worktree directory names that reflect the branch or purpose (e.g., `feature-login-wt`, `hotfix-prod-debug`, `pr-123-review`).

Advanced Techniques: Team Integration & Complex Scenarios

Integrating worktrees into team workflows primarily focuses on local developer efficiency. While worktrees themselves aren't shared across a network, they enable individual developers to manage multiple contexts against a shared remote more effectively.

Debugging Production Issues: Quickly check out a stable release branch to reproduce and debug issues without affecting your current development work.

```
# In main_project (on your current feature branch)
# A critical bug is reported on the 'release/v1.2' branch.
git worktree add ../prod-debug-v1.2 release/v1.2
# Navigate to ../prod-debug-v1.2, reproduce the bug, develop a fix, and commit.
# Then, cherry-pick or merge the fix back to 'main' and 'release/v1.2' as
appropriate.
```

Temporary Experimental Branches: Use worktrees for throwaway experiments that don't warrant a full branch in the main repo, or to test a new library version without polluting your main development environment.

```
# In main_project (on 'main')
# Experiment with a new library on a temporary branch 'exp-new-lib'
git worktree add --detach ../experiment-new-lib
# Expected output:
# Preparing working tree (detached HEAD)
# HEAD is now at 8e3c0f2 Initial commit
#
# In ../experiment-new-lib, you are in a detached HEAD state.
# You can commit here, or create a new branch if the experiment is successful.
# Once done, simply remove the worktree:
# cd ../experiment-new-lib
# git worktree remove . # Removes the current worktree
# rm -rf ../experiment-new-lib
```

The `--detach` flag creates a worktree in a detached HEAD state, useful for quick, isolated experiments.

Quick Reference

- **`git worktree add <path> [branch]`**: Creates a new worktree at `<path>` checking out `[branch]`. If `[branch]` doesn't exist, a new branch is created. If omitted, `HEAD` is checked out.
- **`git worktree add --detach <path>`**: Creates a worktree in a detached HEAD state.
- **`git worktree list`**: Shows all active worktrees.
- **`git worktree remove <path>`**: Removes the worktree at `<path>`. The directory must be empty or Git will error.
- **`git worktree prune`**: Cleans up stale worktree references (e.g., if a worktree directory was manually deleted).
- **Internal Model**: Worktrees share `.git/objects` but each has its own `.git/HEAD`, `.git/index`, and `.git/worktree` file pointing to its specific state and back to the main repository.
- **Gotcha**: A branch can only be checked out in one worktree at a time.

References

1. Official Git Documentation - git-worktree
2. Git SCM Book - Git Tools - Worktrees
3. Atlassian Git Tutorial - Git Worktree

This page is AI-assisted. References official documentation.

CHAPTER 05

LangChain Catalyst - LLM Orchestration Essentials

LangChain v0.2.x (Jan 2026 release cycle), Python 3.10+

Core Syntax

Instantiate a ChatModel and get a basic completion. Ensure `OPENAI_API_KEY` is set in your environment.

```
from langchain_openai import ChatOpenAI
# Modern practice: specific integration imports
from langchain_core.messages import HumanMessage # Standard message types

# Initialize a chat model. Default model is typically gpt-3.5-turbo.
llm = ChatOpenAI(temperature=0.7) # Adjust creativity (0.0-1.0)

# Invoke the model with a simple message.
response = llm.invoke([
    HumanMessage(content="What is the capital of France?")
# Input as a list of messages
])

print(response.content) # Access the generated text content
```

Essential Patterns

Combine prompts and models using LangChain Expression Language (LCEL) for robust, composable chains.

```
from langchain_core.prompts import ChatPromptTemplate
from langchain_openai import ChatOpenAI
from langchain_core.output_parsers import StrOutputParser
# Parses model output to a string

# Define a chat prompt template with a system and human message.
prompt = ChatPromptTemplate.from_messages([
    ("system", "You are a helpful AI assistant. Respond concisely."), # System
instructions
    ("user", "{question}") # User input placeholder
])

# Create a simple chain: Prompt -> Model -> Output Parser.
# The '|' operator pipes the output of one component as input to the next.
chain = prompt | ChatOpenAI(temperature=0.3) | StrOutputParser()

# Invoke the chain with input variables.
response = chain.invoke({"question": "What is the largest ocean on Earth?"})

print(response)
```

Retrieval Augmented Generation (RAG)

Integrate external knowledge by retrieving relevant documents and passing them to the LLM.

```
import os
from langchain_community.document_loaders import TextLoader # Load documents
from files
from langchain_text_splitters import RecursiveCharacterTextSplitter # Split
text into chunks
from langchain_openai import OpenAIEmbeddings # Generate vector embeddings
from langchain_community.vectorstores import FAISS # In-memory vector store
from langchain_core.runnables import RunnablePassthrough
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate

# 1. Load and split documents
loader = TextLoader("data/example.txt")
# Assume 'data/example.txt' exists with text
docs = loader.load()
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=2
00)
splits = text_splitter.split_documents(docs)

# 2. Create embeddings and vector store
embeddings = OpenAIEmbeddings()
vectorstore = FAISS.from_documents(splits, embeddings) # Create index from
splits and embeddings
retriever = vectorstore.as_retriever() # Convert vector store to a retriever

# 3. Define RAG prompt
rag_prompt = ChatPromptTemplate.from_messages([
    ("system", "Answer the question based ONLY on the following context:\n{cont
ext}"),
    ("user", "{question}")
])

# 4. Construct RAG chain
# RunnablePassthrough allows passing input through, optionally mapping it.
# 'context' key is populated by the retriever.
rag_chain = (
    {"context": retriever, "question": RunnablePassthrough()} # Map input to
retriever and question
    | rag_prompt
    | ChatOpenAI(temperature=0)
    | StrOutputParser()
)

# Invoke the RAG chain.
# Ensure 'data/example.txt' contains information about "Python" for a
meaningful response.
print(rag_chain.invoke("What is Python primarily used for?"))
```

Tools and Agents

Allow LLMs to interact with external systems or perform actions, extending their capabilities.

```python
from langchain.agents import create_react_agent, AgentExecutor
from langchain_core.tools import tool # Modern decorator for defining tools
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate

# Define a custom tool using the @tool decorator.
@tool
def get_current_weather(location: str) -> str:
    """Returns the current weather for a given location."""
    if "london" in location.lower():
        return "It's 10°C and cloudy in London."
    elif "paris" in location.lower():
        return "It's 15°C and sunny in Paris."
    else:
        return "Weather data not available for this location."

# List of tools available to the agent.
tools = [get_current_weather]

# Define the agent's prompt. ReAct agents require specific formatting.
# LangChain provides default prompts for common agent types.
agent_prompt = ChatPromptTemplate.from_messages([
    ("system", "You are a helpful assistant that can use tools."),
    ("user", "{input}"),
    ("placeholder", "{agent_scratchpad}") # Essential for ReAct agent internal
monologue
])

# Initialize the LLM for the agent's reasoning.
llm = ChatOpenAI(temperature=0)

# Create a ReAct agent.
agent = create_react_agent(llm, tools, agent_prompt)

# Create an AgentExecutor to run the agent.
agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True) #
verbose for tracing agent steps

# Invoke the agent.
agent_executor.invoke({"input": "What is the weather in London?"})
```

Memory

Maintain conversational context across turns for stateful interactions.

```python
from langchain.chains import ConversationChain # Simple chain for
conversational memory
from langchain_openai import ChatOpenAI
from langchain.memory import ConversationBufferMemory # Stores conversation
history directly

# Initialize LLM and memory.
llm = ChatOpenAI(temperature=0.5)
memory = ConversationBufferMemory() # Default memory stores all messages

# Create a conversation chain with memory.
# The ConversationChain automatically handles prompt formatting for memory.
conversation = ConversationChain(
    llm=llm,
    memory=memory,
    verbose=True # See the prompt being constructed with memory
)

# First turn
conversation.invoke("My favorite color is blue.")

# Second turn, the LLM remembers the previous statement.
response = conversation.invoke("What is my favorite color?")

print(response['response'])
```

Gotchas & Best Practices

1. Environment Variables: Always manage API keys securely via environment variables.

```python
import os
# Recommended: Set this before running any LangChain code that uses OpenAI.
# os.environ["OPENAI_API_KEY"] = "sk-YOUR_ACTUAL_KEY"
# For production, use a secrets manager or .env files (with dotenv).
```

2. LCEL for Composability & Performance: Prefer LCEL (`|`) over older `Chain` classes for better streaming, batching, and async support.

```python
from langchain_core.prompts import ChatPromptTemplate
from langchain_openai import ChatOpenAI
from langchain_core.output_parsers import StrOutputParser

# BAD: Older, less flexible chain composition
# from langchain.chains import LLMChain
# prompt = ChatPromptTemplate.from_template("Tell me a {adjective} joke.")
# llm_chain = LLMChain(prompt=prompt, llm=ChatOpenAI())
# llm_chain.invoke({"adjective": "funny"})

# GOOD: LCEL for modern, efficient chaining
chain = (
    ChatPromptTemplate.from_template("Tell me a {adjective} joke.")
    | ChatOpenAI(temperature=0.8)
    | StrOutputParser()
)
print(chain.invoke({"adjective": "funny"}))
```

3. When to use LangChain vs. Plain LLM Calls: - **Use LangChain when:** - You need complex orchestration (multiple steps, conditional logic). - Integrating with external data sources (RAG). - Managing conversational memory. - Using tools/agents for dynamic actions. - Experimenting with different LLMs/components. - Requiring robust logging, tracing, and observability (LangSmith). - **Use plain LLM API calls when:** - Simple, single-turn prompts without context or external data. - Extreme latency requirements where framework overhead is critical. - Direct control over every API parameter is paramount.

4. Token Management: Be mindful of context window limits and cost. Use `RecursiveCharacterTextSplitter` with appropriate `chunk_size` and `chunk_overlap` for RAG. Monitor token usage with callbacks or LangSmith.

Advanced Techniques

Leverage async, streaming, and batching for performance and responsiveness.

```python
import asyncio
from langchain_core.prompts import ChatPromptTemplate
from langchain_openai import ChatOpenAI
from langchain_core.output_parsers import StrOutputParser

llm = ChatOpenAI(temperature=0.7)
prompt = ChatPromptTemplate.from_template("Tell me a short story about {topic}.
")
chain = prompt | llm | StrOutputParser()

# Asynchronous invocation for non-blocking operations
async def async_invoke_example():
    response = await chain.ainvoke({"topic": "a space cat"})
    print(f"Async response: {response}\n")

# Streaming for real-time output (e.g., in a UI)
async def stream_example():
    print("Streaming response:")
    async for chunk in chain.astream({"topic": "a magical forest"}):
        print(chunk, end="", flush=True)
    print("\n")

# Batching for parallel processing of multiple inputs
async def batch_example():
    inputs = [{"topic": "a robot chef"}, {"topic": "an underwater city"}]
    responses = await chain.abatch(inputs)
    print(f"Batch responses: {responses}\n")

# Run all examples
asyncio.run(async_invoke_example())
asyncio.run(stream_example())
asyncio.run(batch_example())
```

Quick Reference

- **LCEL Operators**: `|` (pipe), `.invoke()`, `.ainvoke()`, `.stream()`, `.astream()`, `.batch()`, `.abatch()`
- **Core Components**: `ChatOpenAI`, `ChatPromptTemplate`, `StrOutputParser`, `RunnablePassthrough`
- **Retrieval**: `TextLoader`, `RecursiveCharacterTextSplitter`, `OpenAIEmbeddings`, `FAISS`, `.as_retriever()`
- **Agents**: `@tool` decorator, `create_react_agent`, `AgentExecutor`
- **Memory**: `ConversationBufferMemory`, `ConversationChain`
- **Environment**: `OPENAI_API_KEY` (required for OpenAI models)
- **Version Note**: LangChain `v0.2.x` emphasizes LCEL (`langchain_core.runnables`) and specific integrations (`langchain_openai`, `langchain_community`). Older `langchain` package components are being refactored or deprecated.

References

1. LangChain Official Documentation
2. LangChain Expression Language (LCEL) Guide
3. OpenAI API Documentation

This page is AI-assisted. References official documentation.

CHAPTER 06

Monorepo Mastery: npm Workspaces & npx Unlocked

Native monorepo management with npm workspaces and on-demand package execution with npx. Node.js v22.x, npm v10.x (as of 2026-02-10).

Core Setup: npm Workspaces

Initialize a monorepo and define workspace roots in the root `package.json` to enable npm's native monorepo capabilities.

```
// root/package.json
{
  "name": "my-monorepo",
  "version": "1.0.0",
  "private": true, // Prevents accidental publishing of the root package
  "workspaces": [ // Defines directories containing workspace packages
    "packages/*", // Example: packages/ui-lib, packages/utils
    "apps/*"      // Example: apps/web, apps/admin
  ],
  "scripts": {
    "build": "npm run build --workspaces", // Runs 'build' script in all
workspaces
    "test": "npm test --workspaces"        // Runs 'test' script in all
workspaces
  },
  "devDependencies": {
    "typescript": "^5.3.3" // Common dev dependencies are often hoisted to the
root
  }
}
```

Each workspace package also has its own `package.json`.

```
// packages/ui-lib/package.json
{
  "name": "@my-monorepo/ui-lib", // Scoped package name for internal usage
  "version": "1.0.0",
  "main": "dist/index.js", // Entry point after build
  "scripts": {
    "build": "tsc" // Example build script for TypeScript
  },
  "dependencies": {
    "react": "^18.2.0" // UI library specific dependencies
  }
}
```

Local Package Linking & Consumption

An application within the monorepo consumes a shared UI library from another workspace using the `workspace:` protocol.

```
// apps/web/package.json
{
  "name": "@my-monorepo/web",
  "version": "1.0.0",
  "private": true, // Often true for applications not meant for publishing
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build"
  },
  "dependencies": {
    "@my-monorepo/ui-lib": "workspace:*", // Links to the local ui-lib package
    "react": "^18.2.0",
    "react-dom": "^18.2.0"
  }
}
```

The application can then import components directly from the linked workspace package.

```
// apps/web/src/App.js
import React from 'react';
import { Button } from '@my-monorepo/ui-lib'; // Import directly from the
workspace package

function App() {
  return (
    <div>
      <h1>My Web App</h1>
      <Button onClick={() => alert('Clicked!')}>Click Me</Button>
    </div>
  );
}

export default App;
```

Dependency Hoisting & Resolution

`npm install` at the monorepo root intelligently manages dependencies, hoisting common ones to the root `node_modules` to optimize disk space and installation time.

```
# Execute from the monorepo root directory
npm install
# npm analyzes all workspace package.json files.
# Dependencies with compatible versions (e.g., 'react') are hoisted
# to the root 'node_modules' to avoid duplication.
# Unique or conflicting versions remain in their respective workspace's
'node_modules'.
# 'workspace:*' dependencies are automatically symlinked by npm,
# pointing to the actual local package directory.
```

Shared UI Library Development

Developing a reusable React component library and making it available to other applications within the monorepo.

```
// packages/ui-lib/src/Button.js
import React from 'react';

const Button = ({ children, onClick }) => { // Simple functional component
  return (
    <button
      style={{ padding: '10px 20px', backgroundColor: '#007bff', color:
'white', border: 'none', borderRadius: '5px', cursor: 'pointer' }}
      onClick={onClick}
    >
      {children}
    </button>
  );
};

export default Button;
```

```
// packages/ui-lib/src/index.js
export { default as Button } from './Button'; // Export components for
consumption
```

Gotchas & Best Practices: Build Order

Shared packages must be built before any dependent applications. This ensures that the consuming application finds the compiled output when building itself.

```
// root/package.json (extended scripts for build orchestration)
{
  "name": "my-monorepo",
  "version": "1.0.0",
  "private": true,
  "workspaces": [
    "packages/*",
    "apps/*"
  ],
  "scripts": {
    "build:ui-lib": "npm run build --workspace=@my-monorepo/ui-lib", // Build
specific workspace
    "build:web": "npm run build --workspace=@my-monorepo/web",
    "build:all": "npm run build:ui-lib && npm run build:web", // Enforce
correct build order
    "start:web": "npm start --workspace=@my-monorepo/web",
    "lint": "npm run lint --workspaces" // Run linting across all workspaces
  },
  "devDependencies": {
    "lerna": "^8.0.0", // Optional: Lerna can assist with versioning and
publishing workflows
    "typescript": "^5.3.3"
  }
}
```- **Versioning**: Use `workspace:*` for local dependencies to automatically link to the current version within the monorepo. For external packages, use exact versions or caret/tilde ranges.

- **`private: true`**: Set `private: true` in `package.json` for packages not intended for public npm registry.

Advanced Pattern: npx for Temporary Tools

`npx` (Node Package eXecutor) runs npm package binaries without explicit installation, ideal for one-off commands or specific tool versions.

```bash
# Execute a package binary without installing it globally or locally
npx create-react-app my-new-app --template typescript
# npx temporarily downloads 'create-react-app' and its dependencies,
# executes the binary, then removes them.

# Run a specific version of a tool
npx eslint@8.56.0 . --fix
# Ensures a particular ESLint version is used, regardless of local install.

# Execute a binary from a local 'node_modules/.bin' (similar to npm run)
# If 'eslint' is a devDependency, 'npm run eslint' is common.
# 'npx eslint' also works and will find the local binary first.
npx eslint .
# Prioritizes local installation, then temporary download if not found.

# Security note: Always verify packages before executing with npx,
# especially from unknown sources, as it runs arbitrary code.
```

Advanced Pattern: CI/CD with GitHub Actions

Configuring GitHub Actions for monorepos with workspaces requires specific caching and explicit build steps.

```yaml
# .github/workflows/ci.yml

name: Monorepo CI

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4 # Use the latest stable checkout action

      - name: Setup Node.js
        uses: actions/setup-node@v4 # Use Node.js v22.x for 2026
        with:
          node-version: '22'
          cache: 'npm' # Enable npm cache
          cache-dependency-path: '**/package-lock.json' # Cache based on lock files

      - name: Install dependencies
        run: npm install # Installs all dependencies across workspaces, leveraging hoisting

      - name: Build UI Library
        run: npm run build --workspace=@my-monorepo/ui-lib # Explicitly build shared library first

      - name: Build Web Application
        run: npm run build --workspace=@my-monorepo/web # Then build the application

      - name: Run tests
        run: npm test --workspaces # Run tests across all workspaces
```

`- **`actions/checkout@v4`**: Ensures the repository is available.

- **`actions/setup-node@v4`**: Sets up Node.js. `cache: 'npm'` and `cache-dependency-path` are crucial for performance, caching `node_modules` based on `package-lock.json` changes.
- **`npm install`**: Run once at the root to install all workspace dependencies.
- **Explicit Build Order**: Separate steps for building shared libraries and applications ensure correct dependency resolution.
- **`npm test --workspaces`**: Runs tests in all defined workspaces.

Quick Reference

- **npm Workspaces**: Native monorepo solution (npm v7+).
- **`private: true`**: In root `package.json` and unpublishable workspaces.
- **`workspaces` array**: Defines paths to workspace packages in root `package.json`.
- **`workspace:*`**: Dependency range for linking local packages.
- **Hoisting**: Common dependencies move to root `node_modules`.
- **`npm install`**: Run at root to install all workspace dependencies.
- **`npm run <script> --workspace=<name>`**: Run script in a specific workspace.
- **`npm run <script> --workspaces`**: Run script in all workspaces.
- **npx**: Executes package binaries without persistent installation.
- **Use for**: One-off commands, specific tool versions, local binary execution.
- **Performance**: Can be slower due to download, but avoids global pollution.
- **Security**: Verify packages, as `npx` executes arbitrary code.
- **CI/CD**: Cache `node_modules`, ensure correct build order (shared libs first).
- **Common Error**: "Cannot find module" if shared library not built before app build.

References

1. npm Workspaces Official Docs
2. npm-install Official Docs
3. npx Official Docs
4. GitHub Actions setup-node

This page is AI-assisted. References official documentation.

CHAPTER 07

Redis Velocity - Data Store Essentials

Redis is an open-source, in-memory data structure store, used as a database, cache, and message broker. Current stable release is Redis 7.2.x, with 7.4.x in release candidate as of late 2025.

Core Syntax

Basic key-value operations for strings, the simplest data type.

```
SET user:1:name "Alice" EX 3600 NX // Set key 'user:1:name' to "Alice", expire
in 3600 seconds, only if key does NOT exist.
GET user:1:name                 // Retrieve the value associated with
'user:1:name'.
DEL user:1:name                 // Delete the key 'user:1:name'.
INCR page:views                 // Increment the integer value of 'page:views'
by one. Creates key with 0 if non-existent.
DECRBY product:stock 5          // Decrement the integer value of
'product:stock' by five.
```

Essential Patterns

Redis offers diverse data structures. Leverage them for efficient data modeling beyond simple strings.

Hashes: Storing Objects

Represent objects with multiple fields.

```
HSET user:2 id 2 name "Bob" email "bob@example.com" // Set multiple fields for
hash 'user:2'.
HGET user:2 name                                    // Retrieve a single field
'name' from hash 'user:2'.
HGETALL user:2                                      // Retrieve all fields and
values from hash 'user:2'.
HDEL user:2 email                                   // Delete the 'email' field
from hash 'user:2'.
```

Lists: Ordered Collections

Implement queues, stacks, or capped collections.

```
LPUSH tasks:queue "task_A" "task_B" // Push elements to the left (head) of
'tasks:queue'.
RPUSH logs:stream "log_entry_1"     // Push elements to the right (tail) of
'logs:stream'.
LPOP tasks:queue                    // Remove and return the first element from
'tasks:queue'.
RPOP logs:stream                    // Remove and return the last element from
'logs:stream'.
LRANGE logs:stream 0 -1             // Get all elements from 'logs:stream'
(from index 0 to last).
LTRIM logs:stream 0 99              // Keep only the first 100 elements in
'logs:stream', effectively capping it.
```

Sets: Unique, Unordered Collections

Ideal for tracking unique items or relationships.

```
SADD users:online "user_X" "user_Y" // Add members to the set 'users:online'.
Duplicates are ignored.
SMEMBERS users:online               // Get all members of 'users:online'.
SISMEMBER users:online "user_X"     // Check if "user_X" is a member of
'users:online'. Returns 1 or 0.
SREM users:online "user_Y"          // Remove "user_Y" from 'users:online'.
SINTER users:online users:premium   // Get members common to both sets.
```

Sorted Sets: Ordered Unique Collections

Store members with scores, enabling ranking and range queries.

```
ZADD leaderboard 100 "player_A" 200 "player_B" 150 "player_C" // Add members
with scores to 'leaderboard'.
ZSCORE leaderboard "player_B"                                // Get the score of
"player_B".
ZRANGE leaderboard 0 -1 WITHSCORES                           // Get all members
and their scores, ordered by score ascending.
ZREVRANGE leaderboard 0 9 BYSCORE WITHSCORES                 // Get top 10
members by score descending.
ZINCRBY leaderboard 50 "player_A"                            // Increment
"player_A"'s score by 50.
```

Common Use Cases

Redis excels in specific application patterns due to its speed and data structures.

Caching Layer

Store frequently accessed data to reduce database load.

```
// Application logic:
// 1. Try to GET product:123 from Redis.
// 2. If not found, fetch from primary DB.
// 3. SET product:123 with data, and an appropriate expiration.

SET product:123 '{"id":123,"name":"Widget","price":19.99}' EX 300 // Cache
product data for 5 minutes.
GET product:123                                                   // Retrieve
cached product data.
```

Rate Limiting

Implement request throttling using `INCR` and `EXPIRE`.

```
// Example: Allow 10 requests per user per minute.
// Key: 'rate_limit:user:{userId}:{timestamp_minute}'

INCR user:123:req:202512301030 // Increment request count for user 123 in
current minute.
EXPIRE user:123:req:202512301030 60 // Set expiration for 60 seconds (if not
already set).
GET user:123:req:202512301030  // Check current request count. If > 10, deny
request.
```

Publish/Subscribe Messaging

Decouple services with real-time message broadcasting.

```
// Terminal 1 (Subscriber):
SUBSCRIBE chat:room:general // Subscribe to messages on 'chat:room:general'.

// Terminal 2 (Publisher):
PUBLISH chat:room:general "Hello everyone!" // Publish a message to
'chat:room:general'.
PUBLISH chat:room:general "New message here." // Another message.
```

Gotchas & Best Practices

Avoid common pitfalls and ensure optimal Redis performance and stability.

Avoid `KEYS *` in Production

`KEYS *` is a blocking command and can severely impact performance on large datasets. Use `SCAN` for production.

```
// BAD: Blocks Redis for potentially long periods.
KEYS *

// GOOD: Iterates incrementally, non-blocking. Returns cursor and batch of
keys.
SCAN 0 MATCH user:* COUNT 100 // Start scan from cursor 0, match 'user:*',
return up to 100 keys.
// ... repeat with new cursor until 0 is returned.
```

Set Expiration (TTL) for Temporary Data

Prevent memory exhaustion by always setting `EXPIRE` for cache entries and temporary session data.

```
SET session:user:456 "token_abc" EX 1800 // Store session token for 30 minutes.
TTL session:user:456                    // Check remaining time-to-live for the
key.
PERSIST session:user:456                // Remove expiration from the key.
```

Configure `maxmemory` and Eviction Policy

Essential for managing memory usage, especially when Redis is used as a cache.

```
// In redis.conf or using CONFIG SET
CONFIG SET maxmemory 2gb                  // Limit Redis memory usage to 2GB.
CONFIG SET maxmemory-policy allkeys-lru   // Evict least recently used keys
when maxmemory is reached.
// Other policies: volatile-lru, allkeys-random, volatile-ttl, noeviction, etc.
```

Advanced Techniques

Unlock more powerful Redis capabilities for complex scenarios.

Transactions (MULTI/EXEC)

Ensure atomicity for a sequence of commands. All commands are executed in order, or none.

```
MULTI                               // Start a transaction block.
INCR user:1:balance                 // Command 1: Increment balance.
LPUSH audit:log "user:1 balance updated" // Command 2: Log the update.
EXEC                                // Execute all queued commands atomically.
// If WATCH was used, EXEC will fail if watched keys changed.
```

Lua Scripting

Execute complex, atomic operations directly on the Redis server, reducing network round-trips.

```
// Script to atomically increment a counter and set an expiry if it's the first
increment.
EVAL "local current = redis.call('INCR', KEYS[1]); if current == 1 then
redis.call('EXPIRE', KEYS[1], ARGV[1]); end; return current;" 1 my_counter 60

// KEYS[1] is 'my_counter', ARGV[1] is '60'.
// This script ensures that EXPIRE is set only once when the counter is
initialized to 1.
```

Streams

A powerful, append-only data structure for immutable log-like data, enabling consumer groups.

```
XADD mystream * sensor_id 123 temperature 25.5 // Add an entry to 'mystream',
'*' auto-generates ID.
XREAD COUNT 2 STREAMS mystream 0               // Read 2 entries from
'mystream' starting from ID 0.
XGROUP CREATE mystream mygroup $ MKSTREAM      // Create consumer group
'mygroup' for 'mystream', starting from latest entry.
XREADGROUP GROUP mygroup consumer1 COUNT 1 STREAMS mystream > // Consumer
'consumer1' reads 1 new entry from 'mygroup'.
XACK mystream mygroup 1678881234567-0           // Acknowledge processing of
entry ID 1678881234567-0.
```

Quick Reference

- **Data Types**: Strings, Hashes, Lists, Sets, Sorted Sets, Streams, Geospatial, Bitmaps, HyperLogLogs.
- **Persistence**: RDB (snapshotting), AOF (append-only file). Both can be enabled.
- **High Availability**: Redis Sentinel (for automatic failover), Redis Cluster (for sharding and HA).
- **Transactions**: `MULTI`, `EXEC`, `DISCARD`, `WATCH`.
- **Lua Scripting**: `EVAL`, `EVALSHA`. Atomic execution.
- **Memory Management**: `maxmemory`, `maxmemory-policy`. Crucial for cache use.
- **Monitoring**: `INFO`, `MONITOR`, `CLIENT LIST`.
- **Security**: Bind to specific IPs, enable `requirepass`, use TLS (Redis 6+).

References

1. Redis Official Documentation

2. Redis Commands Reference
3. Redis Best Practices
4. Redis Streams Introduction

This page is AI-assisted. References official documentation.

CHAPTER 08

Pattern Power: Regex Fast-Track - JavaScript & Python Essentials

Regular Expressions (Regex) for text pattern matching. Current versions: Python 3.12, JavaScript (ECMAScript 2024/ES15).

Core Syntax

Regex literals (JS) or compiled patterns (Python) define the search pattern. Flags modify behavior.

```javascript
// JavaScript: Regex literal (preferred for static patterns)
const reLiteral = /abc/i; // Matches "abc" case-insensitively

// JavaScript: RegExp constructor (for dynamic patterns from strings)
const patternString = "xyz";
const reConstructor = new RegExp(patternString, 'g'); // Matches "xyz" globally

// Test method returns boolean
console.log(reLiteral.test("ABC")); // true
console.log(reConstructor.test("0xyz1xyz2")); // true
```

```python
import re

# Python: Compile a regex pattern (preferred for repeated use)
pattern_compiled = re.compile(r"abc", re.IGNORECASE) # Matches "abc" case-insensitively

# Python: Direct function call (for one-off uses)
match_obj = re.search(r"xyz", "0xyz1xyz2", re.M) # Searches for "xyz" in the string

# Match object evaluates to True if a match is found
print(bool(pattern_compiled.search("ABC"))) # True
print(bool(match_obj)) # True
```

Essential Patterns

Character classes simplify matching common types of characters. Quantifiers specify how many times a character or group must appear.

```javascript
// JavaScript: Character classes and quantifiers
const text = "The quick brown fox jumps over 12 lazy dogs.";

// \d+ : one or more digits
console.log(text.match(/\d+/g)); // ["12"]

// \w+ : one or more word characters (alphanumeric + underscore)
console.log(text.match(/\w+/g)); // ["The", "quick", "brown", "fox", "jumps",
"over", "12", "lazy", "dogs"]

// \s+ : one or more whitespace characters
console.log(text.match(/\s+/g)); // [" ", " ", " ", " ", " ", " ", " ", " ", "
"]

// . : any character (except newline)
console.log("a.b".match(/a.b/)); // ["a.b"]

// {n,m} : between n and m occurrences
console.log("aaaaabbb".match(/a{2,4}/)); // ["aaaa"] (greedy by default)
```

```python
import re

text = "The quick brown fox jumps over 12 lazy dogs."

# Python: Character classes and quantifiers
# \d+ : one or more digits
print(re.findall(r"\d+", text)) # ['12']

# \w+ : one or more word characters (alphanumeric + underscore)
print(re.findall(r"\w+", text)) # ['The', 'quick', 'brown', 'fox', 'jumps',
'over', '12', 'lazy', 'dogs']

# \s+ : one or more whitespace characters
print(re.findall(r"\s+", text)) # [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']

# . : any character (except newline)
print(re.search(r"a.b", "a.b").group(0)) # a.b

# {n,m} : between n and m occurrences
print(re.search(r"a{2,4}", "aaaaabbb").group(0)) # aaaa (greedy by default)
```

Common Use Cases

Regex excels at data validation, extraction, and string manipulation like find-and-replace.

```javascript
// JavaScript: Email validation (basic)
const email = "test@example.com";
const invalidEmail = "invalid-email";
const emailPattern = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;

console.log(emailPattern.test(email)); // true
console.log(emailPattern.test(invalidEmail)); // false

// JavaScript: Extracting specific data with capturing groups
const logEntry = "ERROR 2025-12-27 User 'admin' failed login from
192.168.1.100";
const logPattern = /ERROR (\d{4}-\d{2}-\d{2}) User '(\w+)' failed login from
(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})/;
const match = logEntry.match(logPattern);

if (match) {
    console.log(`Date: ${match[1]}, User: ${match[2]}, IP: ${match[3]}`);
}
// JavaScript: Replace all occurrences
const sentence = "The dog chased the cat. The cat ran away.";
console.log(sentence.replace(/the/gi, "a")); // "A dog chased a cat. A cat ran
away."
```

```python
import re

# Python: Email validation (basic)
email = "test@example.com"
invalid_email = "invalid-email"
email_pattern = re.compile(r"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$")

print(bool(email_pattern.match(email))) # True
print(bool(email_pattern.match(invalid_email))) # False

# Python: Extracting specific data with capturing groups
log_entry = "ERROR 2025-12-27 User 'admin' failed login from 192.168.1.100"
log_pattern = re.compile(r"ERROR (\d{4}-\d{2}-\d{2}) User '(\w+)' failed login
from (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})")
match = log_pattern.search(log_entry)

if match:
    # Access groups by index or name
    print(f"Date: {match.group(1)}, User: {match.group(2)}, IP:
{match.group(3)}")

# Python: Replace all occurrences
sentence = "The dog chased the cat. The cat ran away."
print(re.sub(r"the", "a", sentence, flags=re.IGNORECASE)) # "A dog chased a
cat. A cat ran away."
```

Gotchas & Best Practices

Be aware of greedy vs. non-greedy matching, backslash escaping, and catastrophic backtracking.

```javascript
// JavaScript: Greedy vs. Non-greedy quantifiers
const html = "<div><span>Hello</span><span>World</span></div>";

// Greedy: Matches the longest possible string
console.log(html.match(/<span>.*<\/span>/)); // ["<span>Hello</
span><span>World</span>"]

// Non-greedy (add '?'): Matches the shortest possible string
console.log(html.match(/<span>.*?<\/span>/g)); // ["<span>Hello</span>",
"<span>World</span>"]

// JavaScript: Escaping special characters
// To match a literal dot, use \.
console.log("file.txt".match(/file\.txt/)); // ["file.txt"]
console.log("file.txt".match(/file.txt/)); // ["file.txt"] - also matches
"fileXtxt"
```

```python
import re

# Python: Greedy vs. Non-greedy quantifiers
html = "<div><span>Hello</span><span>World</span></div>"

# Greedy: Matches the longest possible string
print(re.search(r"<span>.*</span>", html).group(0)) # <span>Hello</
span><span>World</span>

# Non-greedy (add '?'): Matches the shortest possible string
print(re.findall(r"<span>.*?</span>", html)) # ['<span>Hello</span>',
'<span>World</span>']

# Python: Raw strings for regex patterns
# Use r"" to avoid issues with backslashes being interpreted as escape
sequences by Python string literal parser.
print(re.search(r"\bword\b", "a word is here").group(0)) # word
# print(re.search("\bword\b", "a word is here")) # Warning: \b is backspace
character in string literal

# Python: Catastrophic backtracking example (avoid patterns like (a+)+)
# This pattern is highly inefficient and can lead to ReDoS attacks or timeouts.
# re.match(r"(a+)+b",
"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac") # Will be
very slow or fail
```

Advanced Techniques

Lookaheads and lookbehinds assert conditions without consuming characters. Named groups improve readability.

```javascript
// JavaScript (ES2018+): Named Capturing Groups
const dateString = "2025-12-27";
const datePattern = /(?<year>\d{4})-(?<month>\d{2})-(?<day>\d{2})/;
const matchDate = dateString.match(datePattern);

if (matchDate) {
    console.log(`Year: ${matchDate.groups.year}, Month: ${matchDate.groups.mont
h}, Day: ${matchDate.groups.day}`);
}

// JavaScript: Positive Lookahead (?=...)
// Matches "foo" only if it's followed by "bar" (but "bar" is not included in
the match)
console.log("foobar".match(/foo(?=bar)/)); // ["foo"]
console.log("foobaz".match(/foo(?=bar)/)); // null

// JavaScript: Negative Lookahead (?!...)
// Matches "foo" only if it's NOT followed by "bar"
console.log("foobaz".match(/foo(?!bar)/)); // ["foo"]
console.log("foobar".match(/foo(?!bar)/)); // null
```

```python
import re

# Python: Named Capturing Groups
date_string = "2025-12-27"
date_pattern = re.compile(r"(?P<year>\d{4})-(?P<month>\d{2})-(?P<day>\d{2})")
match_date = date_pattern.search(date_string)

if match_date:
    print(f"Year: {match_date.group('year')}, Month:
{match_date.group('month')}, Day: {match_date.group('day')}")

# Python: Positive Lookahead (?=...)
# Matches "foo" only if it's followed by "bar"
print(re.search(r"foo(?=bar)", "foobar").group(0)) # foo
print(re.search(r"foo(?=bar)", "foobaz")) # None

# Python: Negative Lookahead (?!...)
# Matches "foo" only if it's NOT followed by "bar"
print(re.search(r"foo(?!bar)", "foobaz").group(0)) # foo
print(re.search(r"foo(?!bar)", "foobar")) # None

# Python: Positive Lookbehind (?<=...)
# Matches "bar" only if it's preceded by "foo"
print(re.search(r"(?<=foo)bar", "foobar").group(0)) # bar

# Python: Negative Lookbehind (?<!...)
# Matches "bar" only if it's NOT preceded by "foo"
print(re.search(r"(?<!foo)bar", "bazbar").group(0)) # bar
```

Quick Reference

- **Anchors**: `^` (start), `$` (end), `\b` (word boundary), `\B` (non-word boundary)

- **Quantifiers**: `*` (0+), `+` (1+), `?` (0 or 1), `{n}` (exactly n), `{n,}` (n+), `{n,m}` (n to m), `?` (non-greedy suffix)
- **Character Classes**: `.` (any char except newline), `\d` (digit), `\D` (non-digit), `\w` (word char), `\W` (non-word char), `\s` (whitespace), `\S` (non-whitespace), `[abc]` (any of a,b,c), `[^abc]` (not a,b,c)
- **Groups**: `(...)` (capturing), `(?:...)` (non-capturing), `(?<name>...)` (JS/Python named capturing), `(?P<name>...)` (Python named capturing)
- **Alternation**: `|` (OR)
- **Flags (JS)**: `i` (ignore case), `g` (global), `m` (multiline), `u` (unicode), `s` (dotall)
- **Flags (Python)**: `re.IGNORECASE`, `re.MULTILINE`, `re.DOTALL`, `re.VERBOSE`, `re.ASCII`, `re.UNICODE`
- **JS Methods**: `test()`, `match()`, `matchAll()`, `search()`, `replace()`, `split()`
- **Python Methods (re module)**: `search()`, `match()`, `fullmatch()`, `findall()`, `finditer()`, `sub()`, `split()`

References

1. MDN Web Docs: Regular expressions - JavaScript
2. Python Docs: re — Regular expression operations
3. Regex101.com - Online regex tester and debugger (useful for learning and testing patterns)

This page is AI-assisted. References official documentation.

CHAPTER 09

Sed Streamline - Linux Text Crafting Essentials

GNU sed 4.9 (stable as of 2025-12-30). A stream editor for filtering and transforming text.

Core Syntax

The fundamental operation in `sed` is substitution. This block demonstrates basic text replacement.

```
# Example file content:
# line 1: This is a test.
# line 2: Another test line.
# line 3: Test complete.

# Basic substitution: 's/regexp/replacement/flags'
echo "This is a test." | sed 's/test/example/'
# Replaces 'test' with 'example' on the first match per line.
# Output: This is a example.

# Global substitution (g flag): replaces all occurrences on a line.
echo "test test test" | sed 's/test/ok/g' # Replaces all 'test' with 'ok'.
# Output: ok ok ok

# Case-insensitive substitution (I flag - GNU sed extension).
echo "This Is A Test." | sed 's/test/success/I' # Replaces 'Test' with
'success', ignoring case.
# Output: This Is A success.
```

Essential Patterns

`sed` excels at filtering and deleting lines based on patterns or line numbers.

```
# Example file content (imagine this is 'data.txt'):
# Line 1: Header
# Line 2: Item A: Value 1
# Line 3: Item B: Value 2
# Line 4: Item C: Value 3
# Line 5: Footer

# Deleting specific lines by number.
sed '1d' data.txt # Deletes the first line.
sed '3,5d' data.txt # Deletes lines from 3 to 5 (inclusive).

# Deleting lines matching a pattern.
sed '/Footer/d' data.txt # Deletes any line containing "Footer".

# Printing only specific lines (requires -n to suppress default output).
sed -n '2p' data.txt # Prints only the second line.
sed -n '/Item/p' data.txt # Prints only lines containing "Item".

# In-place editing (use with caution, always backup or test first).
# Creates a backup 'data.txt.bak' and modifies 'data.txt'.
# sed -i.bak 's/Value/DATA/' data.txt
# No backup, directly modifies 'data.txt'.
# sed -i 's/Value/DATA/' data.txt
```

Common Use Cases

Beyond simple replacement, `sed` can insert, append, and transform text based on context.

```
# Appending text after a line matching a pattern.
echo "Start" | sed '/Start/a\
> New line after Start.' # Appends a new line after "Start".
# Output:
# Start
# New line after Start.

# Inserting text before a line matching a pattern.
echo "End" | sed '/End/i\
> New line before End.' # Inserts a new line before "End".
# Output:
# New line before End.
# End

# Replacing an entire line matching a pattern.
echo "Old line content" | sed '/Old line/c\
> Replaced line content.' # Replaces the line containing "Old line" entirely.
# Output: Replaced line content.

# Using different delimiters for substitution (useful when '/' is in pattern/
replacement).
echo "/path/to/file" | sed 's#/path/to/file#/new/path#' # Uses '#' as
delimiter.
# Output: /new/path

# Extracting specific fields using regex groups.
echo "Name: John, Age: 30" | sed -n 's/Name: \([^,]*\), Age: \(.*\)/\1 is \2
years old./p'
# Matches "Name: (group1), Age: (group2)" and prints a formatted string.
# Output: John is 30 years old.
```

Gotchas & Best Practices

`sed` can be powerful but also destructive. Always test and understand its behavior.

```
# Always use -i with a backup extension, or test on copies.
# sed -i.bak 's/error/warning/g' logfile.txt # Creates logfile.txt.bak

# Quoting: Single quotes preserve literal interpretation of special characters.
# Double quotes allow shell variable expansion but require escaping sed's
special chars.
MY_VAR="replacement"
echo "text" | sed "s/text/$MY_VAR/" # Works, $MY_VAR is expanded by shell.
echo "text" | sed 's/text/'"$MY_VAR"'/' # Safer for complex patterns, shell
expands only MY_VAR.

# Be aware of greedy vs. non-greedy matching in regex (GNU sed doesn't directly
support non-greedy).
# To match shortest possible string, avoid '.*' if possible or use character
classes.
echo "<a><b><c>" | sed 's/<.*>//' # Greedy: matches from first '<' to last '>'.
# Output: (empty line)
echo "<a><b><c>" | sed 's/<[^>]*>//g'
# Matches non-'>' characters, effectively non-greedy.
# Output:
```

Advanced Techniques

Leverage `sed`'s hold space for multi-line processing and more complex transformations.

```
# Hold space: 'h' copies pattern space to hold space, 'g' copies hold space to
pattern space.
# 'H' appends pattern space to hold space, 'G' appends hold space to pattern
space.
# 'x' exchanges pattern and hold space.

# Example: Process two lines, then print them in reverse order.
# Input:
# Line 1
# Line 2
sed '
N; # Append next line to pattern space (now holds "Line 1\nLine 2")
s/\(.*\)\n\(.*\)/\2\n\1/; # Swap lines using backreferences
' <<EOF
Line 1
Line 2
EOF
# Output:
# Line 2
# Line 1

# Example: Read a line, store it in hold space, then print it after the next
line.
# Input:
# A
# B
# C
sed '
1h;1d; # On first line: copy to hold space, delete from pattern space.
x; # Exchange pattern and hold space (pattern space now has
'A', hold space has current line).
G; # Append hold space (current line) to pattern space (now 'A\nB').
' <<EOF
A
B
C
EOF
# Output:
# B
# A
# C
# (Note: C is not processed by the x;G block as it's the last line)

# Multi-command scripts using -e or semicolons.
echo "hello world" | sed -e 's/hello/hi/' -e 's/world/there/'
# Output: hi there
echo "hello world" | sed 's/hello/hi/; s/world/there/'
# Output: hi there
```

Quick Reference

- `sed 's/regexp/replacement/flags'`: Basic substitution.
- `g` **flag**: Global replacement on a line.
- `I` **flag (GNU)**: Case-insensitive match.
- `d` **command**: Delete lines.

- `p` **command**: Print lines (often with `-n`).
- `-n` **option**: Suppress automatic printing of pattern space.
- `-i[SUFFIX]` **option**: In-place editing, optionally with backup.
- `a\` **command**: Append text after a line.
- `i\` **command**: Insert text before a line.
- `c\` **command**: Change (replace) an entire line.
- `h`, `H`, `g`, `G`, `x`: Hold space commands for multi-line processing.
- `N` **command**: Append next line of input into the pattern space.
- `P` **command**: Print the first line of the pattern space.
- `b label`: Branch to `label`.
- `t label`: Branch if a substitution has been successful since the last input line was read.
- `&` **in replacement**: Represents the matched `regexp`.
- `\1`, `\2`: Backreferences to captured groups in `regexp`.
- **Delimiters**: Any character can be used instead of `/` (e.g., `s#old#new#`).

References

1. GNU sed manual
2. DigitalOcean - Mastering sed Command in Linux
3. Hostinger - Linux sed command: How to use and practical examples
4. GeeksforGeeks - Sed Command in Linux/Unix With Examples

This page is AI-assisted. References official documentation.