Markdown Compatibility Quick Reference Guide

When "Standard" Markdown Isn't So Standard

Last Updated: October 2025

Overview

Markdown was designed as a simple, universal formatting syntax. In practice, not all applications implement the complete specification—even the "basic" elements. This guide helps you understand where compatibility issues arise and how to work around them.

TL;DR: If you're creating Markdown content for wide distribution, test in your target platforms and have fallback strategies ready.

Elements with Inconsistent Support

These elements are part of the original Markdown specification but may not work reliably across all platforms:

🖼️ **Images**

![alt text](image.jpg)

Common Issues:

- Security policies may block external images
- Some platforms convert images to plain text links
- Mobile applications may restrict image rendering
- Corporate environments often disable image loading

Where This Matters Most:

- Messaging platforms with security restrictions
- Corporate wikis and intranets
- Email clients claiming Markdown support
- Forums with user-generated content

📏 Horizontal Rules

***

___

Common Issues:

- Often ignored or stripped entirely
- Different syntax variations (--- vs *** vs ___) may have different support levels
- Some parsers require blank lines before/after

Where This Matters Most:

- Messaging applications (Slack, Teams, Discord variants)
- Note-taking apps with minimal parsers
- Chat interfaces with Markdown support

💬 Blockquotes

> This is a blockquote

Common Issues:

- May render as plain text without styling
- Nested blockquotes often unsupported
- Some email clients strip the formatting entirely

Where This Matters Most:

- Email platforms with Markdown support
- Lightweight note-taking applications
- Mobile-first messaging apps
- Platforms using custom Markdown parsers

Understanding Implementation Variability

Different platform types implement Markdown differently based on their priorities and constraints:

🔒 Security-Conscious Platforms

Characteristics:

- Intentionally limit rendering of certain elements
- Prioritize user safety over full Markdown feature support
- May strip HTML, images, or external links

Common Restrictions:

- Images only from approved domains
- No inline HTML
- Limited or no link support to external sites
- Stripped or sanitized blockquotes

Examples: Public forums, user-generated content platforms, commenting systems

Your Strategy: Assume minimal support; test before deploying content at scale

---

⚡ Minimal Parser Implementations

Characteristics:

- Implement only core formatting (bold, italic, headers)
- Skip elements requiring complex parsing logic
- Prioritize speed and simplicity over completeness

Common Restrictions:

- No image support
- Missing horizontal rules
- Limited list nesting
- Basic or no blockquote support

Examples: Lightweight mobile apps, embedded editors, real-time collaboration tools

Your Strategy: Stick to the "safest subset" (see below)

🎨 Custom Markdown Variants

Characteristics:

- Implement "Markdown-inspired" syntax
- May add proprietary extensions
- Often deviate from standard behavior

Common Issues:

- Unexpected syntax interpretations
- Elements that work differently than expected
- Mixed support for standard elements

Examples: Proprietary note-taking apps, custom CMSs, platform-specific implementations

Your Strategy: Consult platform-specific documentation; don't assume standard behavior

---

📱 Platform-Specific Restrictions

Characteristics:

- Mobile vs. desktop feature disparities
- API limitations that affect rendering
- Integration constraints with other systems

Common Issues:

- Desktop supports images, mobile doesn't
- Web version differs from app version
- Editor supports more than the rendered output

Examples: Cross-platform applications, web-based editors with mobile apps, API-driven systems

Your Strategy: Test on all target platforms before committing to specific syntax

---

The Safest Subset

When you need maximum compatibility across platforms, use only these elements:

✅ **Nearly Universal Support:**

- **Headings**: # H1, ## H2, ### H3
- **Bold**: **text** (double asterisk more reliable than double underscore)
- **Italic**: *text* (single asterisk more reliable than single underscore)
- **Inline code**: `code`
- **Links**: [text](url) (though external links may be restricted)
- **Unordered lists**: - item (hyphen more reliable than asterisk or plus)
- **Ordered lists**: 1. item

Why These Work:

- Simple to parse
- No security implications
- Core formatting needs
- Implemented even in minimal parsers

Testing Recommendations

Before Deploying Content

1. Create a Test Document

```
# Test Heading

**Bold text** and *italic text*


- List item 1

- List item 2


1. Numbered item
```

2. Another item

`inline code`

[Link text](https://example.com)

> Blockquote test

![Image test](https://example.com/image.jpg)

2. Test In Your Target Environment(s)

- Paste the test document into your platform
- Check each element's rendering
- Note which elements fail or render unexpectedly
- Document any syntax variations that work better

3. Test Across Contexts

- Web interface
- Mobile app
- Desktop application
- Email (if applicable)
- API output (if relevant)

4. Document Your Findings Create a compatibility matrix for your specific use case:

Platform: [Name]

✅ Headings: Yes

✅ Bold/Italic: Yes

✅ Links: Yes

⚠️ Images: External only

❌ Horizontal rules: No

✅ Code: Inline only

...

Workarounds for Common Issues

When Images Don't Work

Option 1: Use text links

[View Image: Project Screenshot](https://example.com/screenshot.jpg)

Option 2: Describe the image

[IMAGE: Dashboard showing Q4 metrics]

Screenshot available in shared drive: /reports/q4-dashboard.png

Option 3: Host on approved platforms

- Some platforms allow images from specific domains (imgur, platform's own CDN)
- Upload to platform's media library when available

When Horizontal Rules Don't Work

Option 1: Use headers as separators

Section One

Content here...

Section Two

Content here...

Option 2: Use visual text separators

---

or

* * *

or

• • •

Option 3: Accept the limitation

- White space alone often provides sufficient visual separation

---

When Blockquotes Don't Render

Option 1: Use emphasis

Quote: "This is the quoted text"

— Author Name

Option 2: Use indentation (if supported)

  This is indented text that may render as distinct

Option 3: Use formatting cues

💬 "This is a quote or important callout"

When Lists Break

Common issues:

- Mixed list markers causing problems
- Inconsistent indentation
- Blank lines breaking list continuity

Best practices:

✅ Good:

- Item 1
- Item 2
  - Nested item (2 spaces)

  - Another nested item

- Item 3

❌ Problematic:

- Item 1

* Item 2 (mixed markers)

  + Nested item (mixed markers, inconsistent indent)

- Item 3 (blank line may break list in some parsers)

Platform Categories: What to Expect

Messaging Platforms

Expect: Limited support, real-time constraints, security restrictions
Usually works: Bold, italic, code, simple lists
Often doesn't work: Images, horizontal rules, complex nesting
Strategy: Use minimal formatting; test with a sample message

Documentation Systems

Expect: Strong support, may have custom extensions
Usually works: Most standard elements plus tables and code blocks
Often doesn't work: Platform may override styling
Strategy: Check documentation for platform-specific syntax

Note-Taking Applications

Expect: Highly variable; ranges from minimal to extensive support
Usually works: Basic formatting, headers, lists
Often doesn't work: Implementation-specific; test your specific app
Strategy: Create test note; assume features may change with updates

---

Content Management Systems

Expect: Usually strong support, but may sanitize certain elements
Usually works: Standard elements, often with extensions
Often doesn't work: Raw HTML, scripts, certain embeds (security)
Strategy: Review CMS documentation for allowed syntax

Email Clients

Expect: Most conservative support; heavy restrictions
Usually works: Very basic formatting only
Often doesn't work: Images, links, most advanced features
Strategy: Use plain text with minimal formatting

Developer Tools (IDEs, Git platforms)

Expect: Excellent support, often with extensions
Usually works: Full standard Markdown plus code-specific features
Often doesn't work: Rare; usually full-featured
Strategy: Use confidently, but check for platform-specific extensions

---

Red Flags: When to Be Cautious

Watch out for these warning signs that suggest limited Markdown support:

🚩 **Platform advertises "Markdown support" without specifying which variant**
→ Test thoroughly before assuming standard behavior

🚩 **Mobile and desktop versions look different**
→ Test on all platforms your users will access

🚩 **Documentation is sparse or missing**
→ Assume minimal support; test everything

🚩 **Platform is security-focused (public forums, user content)**
→ Expect aggressive sanitization; use minimal formatting

🚩 **Recent platform updates changed rendering**
→ Compatibility may shift; maintain test documents

🚩 **Community reports inconsistent behavior**
→ Trust user reports; test edge cases

Best Practices for Cross-Platform Content

1. Start Conservative

Use only the safest subset initially, then expand if testing confirms support

2. Test Early and Often

Don't wait until you've created 50 documents to discover images don't work

3. Document Platform Quirks

Keep notes on which platforms support what; save yourself future testing time

4. Have Fallback Strategies

Always have a plan B for when your preferred syntax doesn't work

5. Separate Content from Presentation

When possible, use platform-agnostic source documents and adapt for specific platforms

6. Version Your Content

Different platforms may need different versions; track which is which

7. Communicate Limitations

If you're creating content for others, document which platforms it's tested on

---

For Developers: Implementation Considerations

If you're building an application that supports Markdown:

Be Explicit About Support Level

❌ "Supports Markdown"
✅ "Supports CommonMark standard with images and horizontal rules disabled for security"

Provide Platform-Specific Documentation

Users shouldn't have to guess what works; document your implementation

Consider Security Implications

- Images: External loading risks, privacy concerns
- Links: Phishing, malicious sites
- HTML: XSS vulnerabilities
- Be transparent about what you restrict and why

Test Against CommonMark Spec

Use the official test suite: https://spec.commonmark.org/

Handle Graceful Degradation

When you don't support an element, fail gracefully (show as plain text, don't break)

Additional Resources

Official Specifications:

- CommonMark: https://commonmark.org/
- Original Markdown: https://daringfireball.net/projects/markdown/

Testing Tools:

- Babelmark 3: Compare rendering across implementations
- CommonMark Test Suite: Validate parser behavior

Related Quick Reference:

- Markdown Flavor Extensions Repository: [Link to your GitHub repo]

When to Revisit This Guide

Markdown implementation landscape changes. Revisit this guide when:

- ✅ Adopting a new platform for your content
- ✅ Users report rendering issues
- ✅ Platform updates change Markdown behavior
- ✅ Expanding content to new audience/platforms

- ✅ Building new features that depend on Markdown
- ✅ Every 6-12 months as ecosystem evolves

Contributing

Found a compatibility issue not covered here? Implementation changed? Have a better workaround?

This guide lives alongside practical Markdown resources for the community.

Remember: Markdown's strength is its simplicity. When in doubt, simpler is better. The goal is communication, not perfect formatting.

When it works: Markdown is elegant and efficient
When it doesn't: Plain text is always an option

Use the right tool for your audience and platform.