Taking Multiline Input in Bash Shell Scripting

You're absolutely right — in my previous guide, I covered reading lines in loops and here-documents briefly, but I **didn't give multiline interactive input the dedicated attention it deserves**. That's a common and very practical use case (e.g., letting a user paste a block of text, enter a multi-line message, or provide config data). Sorry for skipping over it in detail!

Here's a **complete, focused guide** specifically on handling **multiline input** from the user interactively in Bash.

Why Multiline Input Is Tricky

- `read` normally stops at the first newline.
- Users often want to press Enter multiple times without ending input.
- Common patterns: read until Ctrl+D (EOF), until a specific sentinel line (like "." or "END"), or until an empty line.

---

1. Read Until Ctrl+D (EOF) – Most Common & Clean

The user types lines and presses **Ctrl+D** (or Ctrl+Z on Windows sometimes) to signal end-of-input.

```
Bash
#!/bin/bash

echo "Enter multiline text (Ctrl+D to finish):"
cat > temp_input.txt   # reads everything until EOF into a file

# Or store directly in a variable (Bash 4+ recommended)
input=$(cat)           # reads from stdin until EOF

echo "You entered:"
echo "$input"
```

Better variable version (preserves formatting):

```bash
#!/bin/bash

echo "Paste your text (Ctrl+D when done):"

# Use mapfile (Bash 4+) for array, or direct variable
input=$(cat << 'EOF'
$(cat)
EOF
)

# Simplest and most reliable:
input=""
while IFS= read -r line; do
    input+="$line"$'\n'
done

echo "=== Your input ==="
printf '%s' "$input"
```

Recommended robust version (works everywhere):

```bash
#!/bin/bash

echo "Enter multiline text (Ctrl+D to end):"

input=""
while IFS= read -r line || [[ -n "$line" ]]; do   # handles last line without
    input+="$line"$'\n'
done

echo "You entered:"
printf '%s' "$input"
```

- The `|| [[ -n "$line" ]]` part ensures the last line is captured even if no trailing newline.
- `IFS=` prevents leading/trailing whitespace trimming.

2. Read Until a Sentinel Line (e.g., "END" or "." on its own line)

Very common in interactive tools (like mail programs or editors).

```
Bash
#!/bin/bash

echo "Enter multiline message (type END on a line by itself to finish):"

input=""
while IFS= read -r line; do
    if [[ "$line" == "END" ]]; then
        break
    fi
    input+="$line"$'\n'
done

echo "Your message:"
printf '%s' "$input"
```

Variant with empty line as terminator:

```
Bash
echo "Enter text (empty line to finish):"

input=""
while IFS= read -r line; do
    if [[ -z "$line" ]]; then   # empty line
        break
    fi
    input+="$line"$'\n'
done

echo "Done!"
```

Variant with "." (like SMTP or old editors):

```
Bash

if [[ "$line" == "." ]]; then
    break
fi
```

---

3. Using `read -d ''` (Null Delimiter) – Advanced

Reads everything until a NUL byte (rare interactively, but useful programmatically).

Interactively, user would need to input Ctrl+@ (NUL), which is awkward → not recommended for humans.

Better for programmatic use.

---

4. Prompt with Here-Document Style (Visual Feedback)

Simulate an editor-like experience:

```
Bash

#!/bin/bash

cat << 'PROMPT'
Enter your multiline text below.
Type 'DONE' on a line by itself when finished.
------------------------------------------------
PROMPT

input=""
while IFS= read -r line; do
    [[ "$line" == "DONE" ]] && break
    input+="$line"$'\n'
done

echo "=== Final input ==="
printf '%s' "$input"
```

---

5. Store as Array (Line-by-Line Processing)

If you want to process lines individually:

```
Bash

#!/bin/bash

echo "Enter lines (Ctrl+D to end):"

lines=()
while IFS= read -r line || [[ -n "$line" ]]; do
    lines+=("$line")
done

echo "You entered ${#lines[@]} lines:"
for line in "${lines[@]}"; do
    echo "→ $line"
done
```

Or with `mapfile` (Bash 4+ only):

```
Bash

echo "Enter lines (Ctrl+D to end):"
mapfile -t lines   # -t removes trailing newlines
printf 'Line: %s\n' "${lines[@]}"
```

---

6. Real-World Examples

A. Simple Note-Taking Script

```Bash
#!/bin/bash

read -p "Title: " title
echo "Enter note body (Ctrl+D to save):"

note=$(cat)

echo "Saving note: $title"
echo "$note" > "notes/${title}.txt"
echo "Saved!"
```

B. Multi-Line Password or Secret (Hidden Input)

Multiline hidden input is hard (no built-in), but you can do line-by-line hidden:

```Bash
input=""
while IFS= read -r -s line || [[ -n "$line" ]]; do
    if [[ "$line" == "END" ]]; then
        echo   # newline
        break
    fi
    input+="$line"$'\n'
    echo   # visual feedback without showing text
done
```

Quick Comparison Table

	Method		Terminator		Preserves Formatting	Works in Bash 3+	Best For
`while read` + Ctrl+D	Ctrl+D (EOF)	Yes	Yes	Pasting large text			
`while read` + sentinel	Custom word (END)	Yes	Yes	Structured input			
`while read` + empty line	Blank line	Yes	Yes	Casual notes			
`mapfile`	Ctrl+D	Yes (array)	Bash 4+ only	Line-by-line processing			
`cat` into variable	Ctrl+D	Yes	Yes	Simple one-shot capture			

These patterns cover **99% of multiline input needs** in Bash scripts.

If you're building something specific (like a mini editor, config wizard, or chat-like input), share the use case and I'll give you the exact snippet! 😊

Thanks for calling me out — this definitely deserved its own spotlight.