Shredder Concept to Quest

paulr

ABSTRACT

This report documents the design, architecture, and implementation of ‘shredder.sh‘, a POSIX-compliant shell script that automates the publication of Markdown documents to HTML, Gopher text, and PDF formats on SDF.org BSD servers. The script applies a wizard pattern â prompting for missing information through structured shell-function interactions rather than failing silently â and frames the entire pipeline as a *Teenage Mutant Ninja Turtles* quest narrated by Master Splinter via ‘cowsay(1)‘ speech bubbles. Each processing stage is assigned a named character; the main orchestrator function is called ‘SHREDDER‘. The system requires no external runtimes, no package managers, and no configuration files beyond a single variable block at the top of the

Laboratory Report â Automated POSIX Markdown Publishing Pipeline Paul R. â SDF.org â `paulr.sdf.org` â 2026-05-19

This report documents the design, architecture, and implementation of `shredder.sh`, a POSIX-compliant shell script that automates the publication of Markdown documents to HTML, Gopher text, and PDF formats on SDF.org BSD servers. The script applies a wizard pattern â prompting for missing information through structured shell-function interactions rather than failing silently â and frames the entire pipeline as a *Teenage Mutant Ninja Turtles* quest narrated by Master Splinter via `cowsay(1)` speech bubbles. Each processing stage is assigned a named character; the main orchestrator function is called `SHREDDER`. The system requires no external runtimes, no package managers, and no configuration files beyond a single variable block at the top of the script. It targets the NetBSD-based ARPA shell environment at SDF.org. This document is itself the inaugural post of the Shredder Quest on `paulr.sdf.org`, and will be published by the tool it describes.

Public-access Unix systems such as SDF.org provide shell accounts, Gopher hosting, and web hosting with nothing but a terminal and an SSH client. Publishing to all three formats from a single Markdown source file is a three-step process that is easy to forget and tedious to repeat. The goal of `shredder.sh` was to reduce that to one command â or zero arguments, if the document does not yet exist.

A secondary goal emerged during design: the tool should not be silent. Traditional shell pipelines communicate through exit codes and sparse log lines. This script takes the opposite approach: every stage is announced by its assigned character, and the transitions between stages are narrated by Master Splinter, rendered as `cowsay(1)` speech bubbles with a custom ASCII rat cow file generated at runtime.

The tertiary goal â which was not planned â was that the whole thing should be fun to run.

This report documents the design decisions, function architecture, control-flow branches, utility dependencies, and shell-portability rationale. It is simultaneously a technical record and, upon being published by the tool it describes, the first entry in the Shredder Quest on `paulr.sdf.org`.

This mission is yours, if you choose to accept it.

Item	Detail
Service	SDF.org Public Access Unix
Account tier	ARPA (ARPA-level shell access)
Operating system	NetBSD (primary target); any POSIX BSD Unix
Shell	`/bin/sh` â POSIX sh; not bash, not zsh
Access method	SSH to `tty.sdf.org`
Gopher hosting	`gopher.sdf.org/users/$USER`
Web hosting	`sdf.org/~$USER` or `$USER.sdf.org`

SDF.org is a public-access Unix system founded in 1987, operating continuously since. MetaARPA accounts include Gopher server space, personal web hosting, email, and access to the full NetBSD base system. The arpa designation is a reference to ARPANET, the precursor to the modern internet, and to the academic culture of public shell accounts that predates the web.

`shredder.sh` depends exclusively on tools present in the NetBSD base system or otherwise standard on POSIX platforms. No package manager invocation is required.

Utility	Role in Pipeline
`sh`	POSIX shell interpreter; `set -eu` throughout
`awk`	Block-level MarkdownâHTML and Markdownâgroff-ms state machines
`sed`	Inline MarkdownâHTML substitutions; Gopher text stripping
`grep`	Structural validation â heading counts and fence balance
`spell`	BSD `spell(1)` for interactive spell checking
`groff`	PDF typesetting via the `-ms` macro package
`ps2pdf`	PostScript-to-PDF conversion via Ghostscript
`fold`	Line-wrapping Gopher text to 72 columns
`mktemp`	Isolated temp workspace; cleaned by `trap` on `EXIT`
`ed`	Line editor for interactive document creation in no-arg mode
`cowsay`	Splinter's speech bubbles; optional, graceful fallback
`printf`	POSIX-portable output; `echo` is not used
`date`	ISO 8601 datestamps for derived output filenames
`chmod`	File permission verification in the final stage
`mkdir`	Output directory creation

Command-line tools fail in two modes: silently (producing wrong output) or noisily (printing a usage string and exiting with status 1). Neither is satisfying when the tool is used infrequently by one person.

A third mode exists: the *wizard*. The wizard detects what information is missing and asks for it before proceeding. GUI installers use this pattern. Terminal tools rarely do. `shredder.sh` applies the wizard pattern at three points in the pipeline:

1. **No input file**: rather than print usage and exit, the script prompts for a title, derives a filename and slug, seeds the file with a `# Title` heading, displays a Markdown syntax reference and an `ed(1)` quick-reference card, and drops the user into `ed` for composition. Control returns to the pipeline when `ed` exits.

2. **Spell check gate**: if `spell(1)` finds misspellings, the script displays them and prompts `[y/N]` before continuing. This is Raphael's contribution. He disagrees with everything.

3. **Gophermap subtitle**: before inserting the gophermap entry, the script offers the author the opportunity to supply an extended display title. If the subtitle was already provided in the no-arg wizard flow, this prompt is skipped.

The wizard is implemented entirely in shell functions. No external expect, no Python, no curses. Each prompt uses `printf` and `read -r`. The functions share state through script-level variables â `$SUBTITLE` being the primary example of a variable set in one branch and consumed in a later stage.

The mapping of pipeline stages to *Teenage Mutant Ninja Turtles* characters is structural, not merely cosmetic. The franchise centres on a group of specialists directed by a wise sensei toward a common goal, repeatedly encountering named antagonists along the way before facing the main villain. The pipeline has exactly this shape.

```
| Character | Stage | Task |
| Splinter | Narrator | Introduces each stage; guides the user |
| April O'Neil | Logging (`APRIL_*`) | Chronicles every step with `==>` lines |
| Splinter | Stage 1 | Structural validation â a master checks personally |
| Raphael | Stage 2 | Spell check â the sceptic who questions every word |
| Leonardo | Stage 3 | HTML assembly â disciplined structure |
| Donatello | Stages 3 & 5 | AWK converters â the technologist behind the scenes |
| Michelangelo | Stage 4 | Gopher ASCII â strips all formality into plain text |
| Krang | Stage 5 | PDF via groff â the disembodied brain applies typesetting |
| Bebop | Stage 6 | Gophermap update â chaotic energy, correctly directed |
| Rocksteady | Stage 7 | Sitemap regeneration â methodical, one entry at a time |
| Baxter Stockman | Stage 8 | Permission verification â compound eyes miss nothing |
| Shredder | Orchestrator | The villain who makes everything happen |
```

Splinter is rendered as a 12-line ASCII rat in a custom `cowsay(1)` cow file. The cow file is generated at runtime into the script's temp directory (`$TMP/rat.cow`). No external file is required; the art is embedded as a shell heredoc with backslashes doubled for Perl heredoc interpolation rules.

`shredder.sh` is a single POSIX sh file. All logic â validation, conversion, PDF generation, gophermap management, sitemap regeneration, permission checking, narration â lives in one file. There are no sourced libraries, no helper scripts, no configuration files. Site-specific parameters (hostnames, paths, URLs) are declared as a single variable block near the top and are the only values a new user needs to edit before first use.

The following conditions must hold before the pipeline will run to completion:

1. The input file exists and contains at least one `# Title` line. 2. The HTML template exists at `~/html/template.html` (overrideable with the `-t` flag). 3. If `~/gopher/gophermap` exists, it contains a line reading exactly `#top`. If the file does not exist, the script creates a skeleton. 4. All utilities listed in Â§2.2 are present on `$PATH`. 5. The user has write access to `~/html/` and `~/gopher/`.

The script validates preconditions 1 and 2 explicitly before any stage runs. Precondition 3 is checked and remedied inside BEBOP. Preconditions 4 and 5 are assumed â the script targets a known environment.

A single temp directory is created by `mktemp -d /tmp/shredder.XXXXXX` early in the script â before argument parsing â so that the `rat.cow` file is available for the first Splinter narration. A `trap` on `EXIT`, `INT`, and `TERM` removes the entire directory on script termination. All intermediate files (HTML body fragments, groff source, gophermap scratch files) are written into this directory and cleaned up automatically.

Given an input file with heading `# My Post` run on 2026-05-19:

Artefact	Path
HTML page	`~/html/2026-05-19-My-Post.html`
Gopher text	`~/gopher/2026-05-19-My-Post.txt`
PDF (Gopher)	`~/gopher/2026-05-19-My-Post.pdf`
PDF (web)	`~/html/2026-05-19-My-Post.pdf`
Gophermap	`~/gopher/gophermap` (updated in place)
Sitemap	`~/html/index.html` (fully regenerated)

The main controller function `SHREDDER()` calls eight stage functions in fixed order. Each must complete before the next begins.

```
SHREDDER()
  â
  ââ 1. SPLINTER()              structural validation
  ââ 2. RAPHAEL()               interactive spell check
  ââ 3. LEONARDO()              HTML assembly
  â      ââ DONATELLO_html_body()    awk block-level converter
  â      ââ DONATELLO_inline_sed()   sed inline converter
  ââ 4. MICHELANGELO()          Gopher ASCII text
  ââ 5. KRANG()                 PDF via groff -ms
  â      ââ DONATELLO_ms_body()      awk groff-ms converter
  ââ 6. BEBOP()                 gophermap entry insertion
  ââ 7. ROCKSTEADY()            sitemap regeneration
  ââ 8. BAXTER_STOCKMAN()       file permission verification
```

Counts `#` heading lines with `grep -c`. Counts ` code-fence lines; verifies the count is even (balanced open and close fences). Calls `APRIL_die` on failure, which prints to stderr and exits with status 1.

Strips Markdown syntax with a `sed` pass into a temp file, then pipes through BSD `spell(1)`. If misspellings are found, they are displayed and the user is prompted to continue (`y`) or abort (`N`). This is the first interactive branch point and the only stage that can abort the pipeline with user consent.

Calls `DONATELLO_html_body` (awk) then `DONATELLO_inline_sed` (sed) to produce a body fragment. Extracts `<style>`, `<header>`, and `<footer>` blocks from the HTML template using awk range patterns (`/<tag>/,/<tag>/`). Assembles the complete HTML page with `printf` and `cat`.

`DONATELLO_html_body` is a 50-line awk program implementing a state machine with three boolean states: `in_code`, `in_table`, and implicit paragraph accumulation. HTML entity escaping (`&`, `<`, `>`) is applied before any structural check so that content characters cannot interfere with emitted tag output.

`DONATELLO_inline_sed` applies four `sed` substitutions for backtick code spans, **bold**, *italic*, and `link` patterns.

A single `sed` pass removes all Markdown syntax. Headings map to proportional leading indentation (H1 none, H2 two spaces, H3 three, H4 four). Inline markup is stripped to plain text. Links show link text only. The result is piped through `fold -s -w 72` for Gopher-compliant 72-column line lengths. `APRIL_rule` provides separator lines.

Calls `DONATELLO_ms_body` (awk) to produce a `groff -ms` source file containing `.TL`, `.AU`, `.AI` macros, an optional `.AB`/`.AE` abstract block (populated from a `## Abstract` section if present), and the full document body. PDF generation is attempted in priority order:

1. `groff -Tps | ps2pdf` â PostScript pipeline via Ghostscript 2. `groff -Tpdf` â native groff PDF output 3. Fallback â the `.ms` source is preserved for manual rendering

`DONATELLO_ms_body` mirrors `DONATELLO_html_body` in structure but emits ms(7) macros. Inline markup is handled by an internal awk function `ms_inline()` using

`match()`/RSTART/RLENGTH loops to convert bold, italic, links, and code spans to groff font-change escapes.

Verifies the `#top` anchor. Creates a skeleton gophermap if none exists. Prompts for an optional extended display title (skipped if `$SUBTITLE` is already set from the no-arg wizard flow). Constructs a four-line entry block and inserts it after `#top` using `sed`'s `r` (read-file) command, ensuring the most recent entry always appears first.

Parses the full gophermap with awk, extracting `TITLE<TAB>BASE` records for all article blocks found after the `#top` anchor. Generates an HTML `<p>` block per article with links to HTML, PDF, and Gopher. Assembles the complete `~/html/index.html` from template blocks and the entry list. The sitemap is fully regenerated on every run; it is not appended to.

Applies `chmod 755` to `~/html/` and `~/gopher/`. Applies `chmod 644` to all output files. Performs a paranoid read-verify by inspecting the world-read bit from `ls -l` output via `awk '{print substr($1,8,1)}'`. Issues `APRIL_warn` messages rather than aborting â the author may be operating in a restrictive umask environment. Reports final status.

The script has four explicit branch points outside the linear stage sequence.

```
argc == 0  â  interactive new-post wizard
              prompt: title
              prompt: subtitle/description
              derive: slug, filename in ~/gopher/
              action: seed file with # Title heading
              display: Markdown syntax reference (one line per construct)
              display: ed(1) quick-reference (1,$n / a / i / c / w / q)
              action: open ed(1) for composition
              continue when ed exits
argc == 1  â  use provided file path directly
argc  > 1  â  show_banner() to stderr + exit 1

spell(1) output empty  â  continue silently
spell(1) finds words   â  display list
                          prompt [y/N]
                          y  â  APRIL_note "Continuing."
                          N  â  APRIL_die "Aborted."

$SUBTITLE non-empty (set in Branch 1)  â  skip prompt, use value
$SUBTITLE empty                        â  prompt for extended title

ps2pdf available AND groff -Tps succeeds  â  PostScript pipeline
else groff -Tpdf succeeds                 â  native groff PDF
else                                      â  save .ms source, APRIL_warn
```

The first line is `#!/bin/sh`. The script is written in POSIX sh throughout. This is a deliberate and meaningful choice for the SDF.org environment.

bash: the dominant Linux shell. Supports all POSIX sh syntax plus many extensions (`[[`, `local`, arrays, `(( ))`). Readers coming from bash will find the script entirely familiar while noting the absence of bashisms. The script will run under bash unchanged.

ksh: the Korn shell, from which much of POSIX sh syntax descends. `ksh93` is present on many BSDs. The script is fully compatible.

ash/dash: the Almquist shell, used as `/bin/sh` on Debian and derivatives. Fast, minimal, strict POSIX. The script targets this level of compatibility.

sh on NetBSD: a `ash`-derived POSIX shell. The primary test target. SDF.org runs NetBSD on its primary systems; all utilities named in Â§2.2 are present in the NetBSD base system.

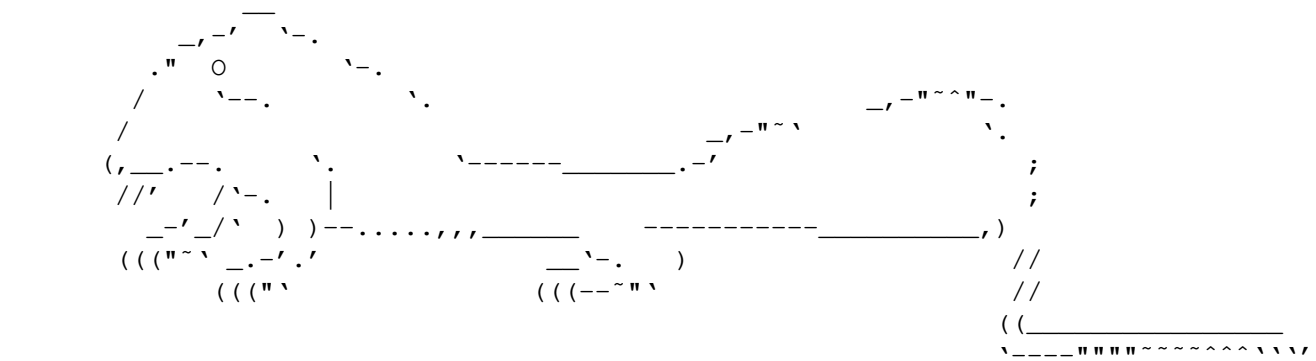
fish: the Friendly Interactive Shell. Entirely incompatible syntax. This script will not run in fish and was not designed to. Fish users are warmly invited to port it.

Specific POSIX choices made throughout:

- `printf` for all output â `echo` is not called anywhere
- `${0##*/}` instead of `$(basename "$0")` â no subshell, no external call
- `$(...)` command substitution â no backticks
- No arrays, no `local`, no `[[`, no `(( ))`, no `+=`
- `read -r` throughout â no backslash processing on input

`SPLINTER_narrate()` pipes any string through `cowsay -f "$RAT_COW" -W 68`. If `cowsay` is absent or the cow file does not exist, output falls back to `~~ Splinter: ... ~~` in plain text. The cow file path is `$TMP/rat.cow`.

The rat cow file is generated by a `cat > "$RAT_COW" << 'RAT_EOF'` heredoc in the script. The shell heredoc uses a single-quoted terminator (`'RAT_EOF'`) to suppress shell variable expansion. The cow file content is a Perl double-quoted heredoc (the cowsay `.cow` format), in which `$thoughts` is the connector variable substituted by cowsay at render time. Literal backslashes in the ASCII art are doubled (`\`) because Perl drops unrecognised escape sequences in double-quoted strings.



Splinter speaks at the following points in execution:

Call	Moment
1	Startup: introduces himself and the quest
2	Startup: describes the cast and the arc
3	After title is known: sizes up the incoming document
4â11	Before each of the 8 pipeline stages
12	Mission complete

Twelve speech bubbles per successful full run.

This file is `2026-05-19-Shredder Concept to Quest.md`. It will be passed to `shredder.sh` as its first argument. Upon completion, the following will exist on the SDF.org server:

```
~/html/2026-05-19-Shredder-Concept-to-Quest.html
~/gopher/2026-05-19-Shredder-Concept-to-Quest.txt
~/gopher/2026-05-19-Shredder-Concept-to-Quest.pdf
~/html/2026-05-19-Shredder-Concept-to-Quest.pdf
~/html/index.html           (updated sitemap, this entry first)
~/gopher/gophermap          (updated, this entry at #top)
```

This is the first post of the Shredder Quest. The index will list it. The gophermap will serve it. The PDF will archive it. The HTML will display it at `http://paulr.sdf.org`.

To run:

```
shredder.sh "2026-05-19-Shredder Concept to Quest.md"
```

Splinter will take it from there.

`shredder.sh` demonstrates that a POSIX shell script can serve as a complete, self-narrating, multi-format publishing pipeline without external runtimes, interpreters, or package ecosystems. The wizard pattern â structured prompts through shell functions in place of bare usage-and-exit â makes the tool accessible for infrequent use without flag memorisation. The TMNT narrative framework transforms a sequence of file transformations into a coherent and memorable terminal experience.

The design is specific to SDF.org NetBSD ARPA shell accounts but is portable to any POSIX system with the utilities named in Â§2.2.

The script is public domain. MIT license. Designed 2026-05-19 by `paulr.sdf.org`.

Cowabunga.

Complete verbatim listing of `shredder.sh` as of the date of this report.

```
#!/bin/sh
# ====================================================================
#  SHREDDER  --  POSIX Markdown Publishing Pipeline
#  "Tonight I dine on turtle soup!"
#  (Tonight I publish a markdown file.  Same energy.)
#
#  Usage:  shredder [-t template.html] [-a "Author Name"] file.md
#
#  Outputs from ~/gopher/ and ~/html/:
#    YYYY-MM-DD-Title.html   styled HTML (html dir)
#    YYYY-MM-DD-Title.txt    72-col ASCII, Gopher (gopher dir)
#    YYYY-MM-DD-Title.pdf    PDF via groff -ms (both dirs)
#    gophermap               updated, newest entry on top
#    index.html              regenerated sitemap (html dir)
#
#  Character Roster:
#    APRIL_*          logging: say / note / warn / die / rule
#    SPLINTER         narrator + stage 1 â structural validation
#    RAPHAEL          stage 2 â interactive spell check
#    DONATELLO_*      awk block converters (html_body, ms_body, inline_sed)
#    LEONARDO         stage 3 â HTML assembly + template injection
#    MICHELANGELO     stage 4 â ASCII/Gopher text conversion
#    FUGITOID         stage 5 â PDF via groff -ms  (defeats KRANG)
#    CASEY_JONES      stage 6 â gophermap regeneration  (defeats BEBOP)
#    MONDO_GECKO      stage 7 â index.html regeneration  (defeats ROCKSTEADY)
#    IRMA             stage 8 â file permission verification  (defeats BAXTER STOCKMAN)
#    SHREDDER         main orchestrator
#
#  Villain roster (errors only â no dialogue):
#    KRANG            android body in the PDF pipeline
#    BEBOP            warthog at the gophermap
#    ROCKSTEADY       rhino at the sitemap
#    BAXTER STOCKMAN  fly-scientist at the permissions gate
#
#  Requires: sed awk spell groff ps2pdf fold mktemp
#  Tested on NetBSD.  POSIX sh throughout.
# ====================================================================

set -eu
PROG="${0##*/}"

# ãã Site configuration ãããããããããããããããããããããããããããããããããããããããããããããããã
# Adjust these for your server before first run.
GOPHER_HOST="gopher.club"
GOPHER_PORT="70"
GOPHER_BASE="/users/${USER}"
HTTP_BASE="http://paulr.sdf.org"
GOPHER_LINK="gopher://gopher.club:70/1/users/paulr/"
SITEMAP_URL="http://paulr.sdf.org/index.html"
MIRROR_BASE=""   # optional HTTP mirror root, e.g. "https://mirror.example.com" â leave bl

# ãã Runtime defaults (overrideable by flags) ãããããããããããããããããããããããããã
TEMPLATE="${HOME}/html/template.html"
AUTHOR="${USER:-anonymous}"
```

```
GOPHERMAP="${HOME}/gopher/gophermap"
WRAP=72
DEFAULT_CATS="deep-thoughts technical-notes idea-incubator lab-reports"


# ======================================================================
# APRIL O’NEIL  â  Logging utilities
# "This is April O’Neil, reporting live from your terminal."
# ======================================================================
APRIL_say()  { printf ’==> %s0 "$*"; }
APRIL_note() { printf ’    %s0 "$*"; }
APRIL_warn() { printf ’%s: WARNING: %s0 "$PROG" "$*" >&2; }
APRIL_die()  { printf ’%s: ERROR: %s0   "$PROG" "$*" >&2; exit 1; }
APRIL_rule() { awk -v n="$WRAP" ’BEGIN{for(i=0;i<n;i++)printf"-";print""}’; }

# ââ Splinter narration â pipes through cowsay rat, falls back to plain text ââ
SPLINTER_narrate() {
    if command -v cowsay >/dev/null 2>&1 && [ -f "$RAT_COW" ]; then
        printf ’%s0 "$*" | cowsay -f "$RAT_COW" -W 68
    else
        printf ’ ~~ Splinter: %s ~~0 "$*"
    fi
    sleep 0.25
}

# ââ Random narration index â varies per call and per run ââââââââââââââââââââââ
# Uses PID * stage-counter so each battle pick differs within one run,
# and differs across runs when the PID changes.
_PID=$$
_NARRATION_SEQ=0
_nidx() {
    _NARRATION_SEQ=$(( _NARRATION_SEQ + 1 ))
    _n=$(( ( _PID + _NARRATION_SEQ * 13 ) % 4 ))
    printf ’%s’ "$_n"
}

show_banner() {
    cat <<BANNER
usage: $PROG [-t template.html] [-a "Author Name"] file.md
       $PROG                     (no args â interactive new-post mode)

  -t  HTML template  (default: ~/html/template.html)
  -a  Author name    (default: SER)

  This will post a markdown file to html, gopher, and pdf on sdf.org
  BSD servers.  Designed 2026-05-19 by paulr.sdf.org
  Public domain â MIT license.
BANNER
}

usage() {
    show_banner >&2
    exit 1
}
```

```
# ââ Temp workspace (early so cow file exists before first narration) âââââââââ
TMP=$(mktemp -d /tmp/shredder.XXXXXX)
trap 'rm -rf "$TMP"' EXIT INT TERM

RAT_COW="$TMP/rat.cow"
cat > "$RAT_COW" << 'RAT_EOF'
##
## Splinter â Master Rat Sensei (TMNT shredder pipeline)
## Backslashes doubled: Perl double-quoted heredoc drops unknown X.
##
$the_cow = <<EOC;
      $thoughts
       $thoughts
           __
       _,-'  `-.
     ."  O       `-.
    /    `--.         `.                              _,-"~^"-.
   /          \         \                    _,-"~`            `.
  (,__.--.     `.        `------_______.-'                      ;
   //'   /`-.   |                                               ;
     _-'_/`  ) )--.....,,,_______\    \-----------_____________,)
   ((("~` _.-'.'               __`-.   )                      //
         (((("`              (((--~"`                        //
                                                            ((_________________
                                                            `----"""""~~~~^^^^\\\\'

EOC
RAT_EOF

show_banner
echo ""

# ââ Ensure default category folders exist under ~/gopher/ âââââââââââââââââââ
for _defcat in $DEFAULT_CATS; do
    mkdir -p "${HOME}/gopher/${_defcat}"
done

SPLINTER_narrate "I am Splinter. Sensei. Master. Rat of considerable wisdom and questionab
echo ""
SPLINTER_narrate "Our allies are ready. April O'Neil will chronicle every step. My student
echo ""

# ââ Argument parsing ââââââââââââââââââââââââââââââââââââââââââââââââââââââââ
while getopts 't:a:h' OPT; do
    case $OPT in
        t) TEMPLATE="$OPTARG" ;;
        a) AUTHOR="$OPTARG"   ;;
        h) usage              ;;
        *) usage              ;;
    esac
done
shift $((OPTIND - 1))

CATEGORY=""   # set by path detection or new-post wizard; used for output routing
```

```
if [ $# -eq 0 ]; then
    # ãã Interactive new-post creation ãããããããããããããããããããããããããããããããããããããã
    echo ""
    SPLINTER_narrate "A student arrives without a manuscript. This is... not entirely surp
    echo ""
    printf '  Post title: '
    read -r TITLE_INPUT
    [ -n "$TITLE_INPUT" ] || APRIL_die "Title cannot be empty."

    _NEW_SLUG=$(printf '%s' "$TITLE_INPUT"         | sed 's/[^A-Za-z0-9 ]//g; s/  */ /g; s
    _NEW_DATE=$(date '+%Y-%m-%d')

    echo ""
    APRIL_say "Available categories (subfolders of ~/gopher/):"
    for _d in "${HOME}/gopher"/*/; do
        [ -d "$_d" ] || continue
        _cn="${_d%/}"; _cn="${_cn##*/}"
        APRIL_note "  $_cn"
    done
    echo ""
    printf '  Category (Enter to skip): '
    read -r CATEGORY
    CATEGORY=$(printf '%s' "$CATEGORY" | tr '[:upper:]' '[:lower:]')
    if [ -n "$CATEGORY" ]; then
        mkdir -p "${HOME}/gopher/${CATEGORY}"
        INPUT="${HOME}/gopher/${CATEGORY}/${_NEW_DATE}-${_NEW_SLUG}.md"
    else
        mkdir -p "${HOME}/gopher"
        INPUT="${HOME}/gopher/${_NEW_DATE}-${_NEW_SLUG}.md"
    fi

    printf '# %s0 "$TITLE_INPUT" > "$INPUT"

    echo ""
    APRIL_note "File: $INPUT"
    echo ""
    APRIL_say "Markdown syntax this pipeline supports:"
    APRIL_note "  # Heading 1"
    APRIL_note "  ## Heading 2"
    APRIL_note "  ### Heading 3"
    APRIL_note "  #### Heading 4"
    APRIL_note "  **bold text**"
    APRIL_note "  *italic text*"
    APRIL_note "  `inline code`"
    APRIL_note "  ```                       fenced code block (open and close with ```)"
    APRIL_note "  [link text](url)"
    APRIL_note "  - list item               (or * list item)"
    APRIL_note "  > blockquote"
    APRIL_note "  ---                       horizontal rule"
    APRIL_note "  | col | col |             pipe-delimited table"
    echo ""
    APRIL_say "ed quick reference:"
    APRIL_note '  1,$n       show all lines with line numbers'
    APRIL_note "  42         go to line 42 (makes it the current line)"
```

```
    APRIL_note "  a          append after current line; end with a lone . on its own line"
    APRIL_note "  i          insert before current line; end with ."
    APRIL_note "             ed carries no indentation â retype all leading spaces yoursel
    APRIL_note "  c          change (replace) current line entirely; end with ."
    APRIL_note "  w          write / save"
    APRIL_note "  q          quit"
    echo ""
    APRIL_say "Typical writing session:"
    APRIL_note "  a          enter append mode (cursor is after the title line)"
    APRIL_note "  <Enter>    blank line after title, then start typing"
    APRIL_note "  Ctrl-V     paste clipboard content (terminal paste)"
    APRIL_note "  <Enter>    finish the last line"
    APRIL_note "  .          lone period on its own line â exits append mode"
    APRIL_note "  w          write (save) the file"
    APRIL_note "  q          quit ed â pipeline resumes automatically"
    echo ""
    SPLINTER_narrate "The scroll has been prepared. Enter ed, write your post, and when fi
    echo ""
    printf '  Press Enter to open ed...'
    read -r _DUMMY
    ed "$INPUT"
    [ -f "$INPUT" ] || APRIL_die "file not found after ed session: $INPUT"
elif [ $# -eq 1 ]; then
    INPUT="$1"
else
    usage
fi

[ -f "$INPUT" ] || APRIL_die "file not found: $INPUT"

# Detect category from path ~/gopher/CATEGORY/file â only if not already set
if [ -z "$CATEGORY" ]; then
    case "$INPUT" in
        "${HOME}/gopher/"*/*)
            _tail="${INPUT#${HOME}/gopher/}"
            CATEGORY=$(printf '%s' "${_tail%%/*}" | tr '[:upper:]' '[:lower:]')
            ;;
    esac
fi

if [ ! -f "$TEMPLATE" ]; then
    SPLINTER_narrate "No template found at $TEMPLATE. A sensei does not leave his students
    mkdir -p "$(dirname "$TEMPLATE")"
    cat > "$TEMPLATE" << 'TMPL_EOF'
<!DOCTYPE html>
<html lang="en">
<head>
<style>
* { margin: 0; padding: 0; box-sizing: border-box; }
body {
  font-family: "Courier New", Courier, monospace;
  font-size: 13px;
  line-height: 1.55;
  color: #000;
```

```
  background: #fff;
  max-width: 80ch;
  margin: 0 auto;
  padding: 2em 1em;
}
header {
  display: flex;
  justify-content: space-between;
  font-weight: bold;
  border-bottom: 1px solid #000;
  padding-bottom: 0.4em;
  margin-bottom: 1.5em;
}
footer {
  display: flex;
  justify-content: space-between;
  font-weight: bold;
  border-top: 1px solid #000;
  padding-top: 0.4em;
  margin-top: 2em;
}
h1 {
  font-size: 13px;
  font-weight: bold;
  text-transform: uppercase;
  margin: 1.6em 0 0.4em 0;
  letter-spacing: 0.04em;
}
h2 {
  font-size: 13px;
  font-weight: bold;
  text-transform: uppercase;
  margin: 1.4em 0 0.3em 0;
  letter-spacing: 0.03em;
}
h3 {
  font-size: 13px;
  font-weight: bold;
  margin: 1.2em 0 0.2em 0;
}
h4 {
  font-size: 13px;
  font-weight: bold;
  margin: 1em 0 0.2em 6ch;
}
p { margin: 0.5em 0 0.5em 6ch; }
li {
  display: block;
  list-style: none;
  margin: 0.4em 0 0.4em 6ch;
}
li::before { content: "- "; }
dt {
  font-weight: bold;
```

```
  margin: 0.8em 0 0 0;
}
dd { margin: 0.1em 0 0.4em 6ch; }
pre {
  margin: 0.5em 0 0.5em 6ch;
  padding: 0.6em 1ch;
  overflow-x: auto;
  border-left: 2px solid #ccc;
  font-size: 12px;
}
code { font-family: "Courier New", Courier, monospace; }
blockquote {
  margin: 0.5em 0 0.5em 8ch;
  padding-left: 2ch;
  border-left: 2px solid #999;
  color: #444;
}
hr {
  border: none;
  border-top: 1px solid #ccc;
  margin: 1em 0;
}
a { color: #000; text-decoration: underline; }
a:hover { background: #000; color: #fff; }
nav {
  margin: 0.5em 0 1.5em 0;
  font-size: 11px;
  color: #555;
  border-bottom: 1px solid #ddd;
  padding-bottom: 0.4em;
}
nav a { margin-right: 2ch; text-decoration: none; }
nav a:hover { background: #000; color: #fff; }
@media print { body { max-width: 100%; } }
</style>
</head>
<body>
<header>
<span>paulr.sdf.org</span>
<span>[COURSE NUMBER] &mdash; IEEE Laboratory Report</span>
<span><a href="index.html">LABREPORT(7)</a></span>
</header>
<footer>
<span>[Institution Name]</span>
<span>LABREPORT(7)</span>
<span>&copy; [YYYY]</span>
</footer>
</body>
</html>
TMPL_EOF
    APRIL_note "Template created: $TEMPLATE"
fi

# ââ Derive output names from date + first # heading âââââââââââââââââââââââââââ
```

```
DATE=$(date '+%Y-%m-%d')
YEAR=$(date '+%Y')

TITLE_RAW=$(sed -n 's/^# //p' "$INPUT" | head -1)
if [ -z "$TITLE_RAW" ]; then
    # No # heading found â derive a title from the filename and prepend it.
    # This lets short posts (even a single paragraph) flow through unchanged.
    _fn="${INPUT##*/}"; _fn="${_fn%.md}"
    case "$_fn" in
        [0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]-*) _fn="${_fn#????-??-??-}" ;;
    esac
    TITLE_RAW=$(printf '%s' "$_fn" | sed 's/[-_]/ /g')
    APRIL_warn "no '# Title' heading â using filename: '$TITLE_RAW'"
    { printf '# %s0 "$TITLE_RAW"; cat "$INPUT"; } > "$TMP/titled.md"
    INPUT="$TMP/titled.md"
fi

SLUG=$(printf '%s' "$TITLE_RAW"     | sed 's/[^A-Za-z0-9 ]//g; s/  */ /g; s/ /-/g'     | t

BASE="${DATE}-${SLUG}"
if [ -n "$CATEGORY" ]; then
    HTML_OUT="${HOME}/html/${CATEGORY}/${BASE}.html"
    PDF_HTML="${HOME}/html/${CATEGORY}/${BASE}.pdf"
    TXT_OUT="${HOME}/gopher/${CATEGORY}/${BASE}.txt"
    PDF_GOPHER="${HOME}/gopher/${CATEGORY}/${BASE}.pdf"
    mkdir -p "${HOME}/html/${CATEGORY}" "${HOME}/gopher/${CATEGORY}"
else
    HTML_OUT="${HOME}/html/${BASE}.html"
    PDF_HTML="${HOME}/html/${BASE}.pdf"
    TXT_OUT="${HOME}/gopher/${BASE}.txt"
    PDF_GOPHER="${HOME}/gopher/${BASE}.pdf"
    mkdir -p "${HOME}/html" "${HOME}/gopher"
fi

SPLINTER_narrate "A markdown file has entered my dojo. Title: '$TITLE_RAW'. I have seen ma
APRIL_say "SHREDDER activated."
APRIL_note "Input  : $INPUT"
APRIL_note "Title  : $TITLE_RAW"
APRIL_note "Base   : $BASE"
APRIL_note "Category: ${CATEGORY:-uncategorized}"
echo ""


# ====================================================================
# SPLINTER  â  Stage 1: Structural Validation
# "A ninja's greatest tool is patience... and awk."
# ====================================================================
SPLINTER() {
    APRIL_say "SPLINTER: Validate"

    # awk avoids the grep -c exit-code trap: grep exits 1 on zero matches,
    # so "grep -c ... || echo 0" can produce "00" instead of "0".
    H1=$(awk '/^# /{n++} END{print n+0}' "$INPUT")
    [ "$H1" -ge 1 ] || APRIL_die "no '# Title' heading found"
```

```
    APRIL_note "$H1 heading(s) found"

    FENCES=$(awk '/^```/{n++} END{print n+0}' "$INPUT")
    if [ $(( FENCES % 2 )) -ne 0 ]; then
        echo ""
        APRIL_warn "odd number of code fences ($FENCES) â one may be unclosed"
        echo "  Fence locations:"
        awk '/^```/{printf "    line %d: %s0, NR, $0}' "$INPUT"
        echo ""
        printf "  Continue anyway? [y/N] "
        read -r _ANS
        case "$_ANS" in
            [Yy]*) APRIL_note "Continuing past fence warning." ;;
            *)     APRIL_die "Aborted. Fix the unclosed fence and re-run." ;;
        esac
    else
        APRIL_note "fences balanced ($FENCES)"
    fi
}


# ====================================================================
# RAPHAEL  â  Stage 2: Interactive Spell Check
# "Maybe I *like* being wrong.  ...No I don't.  Fix it."
# ====================================================================
RAPHAEL() {
    APRIL_say "RAPHAEL: Spell check"

    STRIPPED="$TMP/stripped.txt"

    # Strip markdown syntax before handing off to spell(1).
    # Code blocks are removed entirely; inline markers are erased.
    sed         -e '/^```/,/^```/d'         -e 's/^#1,6  *//g'         -e 's/[1]g'         -e

    # spell(1) may exit non-zero when it finds words; capture safely
    MISSPELLED=$(spell < "$STRIPPED" 2>/dev/null) || true

    if [ -z "$MISSPELLED" ]; then
        APRIL_note "OK â no misspellings found"
        return 0
    fi

    echo ""
    echo "  Possible misspellings:"
    printf '%s0 "$MISSPELLED" | sed 's/^/    /'
    echo ""
    printf "  Continue anyway? [y/N] "
    read -r ANS
    case "$ANS" in
        [Yy]*) APRIL_note "Continuing. (Raphael disagrees.)" ;;
        *)     APRIL_die "Aborted. Correct spelling and re-run." ;;
    esac
}
```

```
# ====================================================================
# DONATELLO  â  AWK Block-Level Converters
# "I have a machine for this.  Obviously."
# Two functions: html_body and ms_body.  Both use the same
# state-machine pattern: escape first, detect structure second.
# A third function handles inline sed substitutions.
# ====================================================================

# HTML block structure.
# Lines are HTML-escaped (&, <, >) before any structural check,
# so content and tag output cannot interfere.
# Paragraphs accumulate in 'para' and are flushed on blank lines
# or block-level elements.
DONATELLO_html_body() {
    awk '
    BEGIN { in_code=0; in_table=0; para="" }

    function flush() {
        if (para != "") { print "<p>" para "</p>"; para="" }
    }
    function escape() {
        gsub(/&/, "\&amp;")
        gsub(/</, "\&lt;")
        gsub(/>/, "\&gt;")
    }

    /^```/ {
        if (!in_code) { flush(); print "<pre><code>"; in_code=1 }
        else          { print "</code></pre>"; in_code=0 }
        next
    }
    in_code { escape(); print; next }

    /^ / {
        if (/^[-: |]+[[:space:]]*$/) next   # skip separator rows
        if (!in_table) { flush(); print "<pre>"; in_table=1 }
        escape(); print; next
    }
    in_table && !/^ / { print "</pre>"; in_table=0 }

    /^#### / { flush(); escape(); sub(/^#### /,""); print "<h4>"$0"</h4>"; next }
    /^### /  { flush(); escape(); sub(/^### /,"");  print "<h3>"$0"</h3>"; next }
    /^## /   { flush(); escape(); sub(/^## /,"");   print "<h2>"$0"</h2>"; next }
    /^# /    { flush(); escape(); sub(/^# /,"");    print "<h1>"$0"</h1>"; next }

    /^---[[:space:]]*$/ { flush(); print "<hr>"; next }

    /^> / {
        flush(); sub(/^> /,""); escape()
        print "<blockquote><p>"$0"</p></blockquote>"; next
    }
    /^[-*] / {
        flush(); sub(/^[-*] /,""); escape()
        print "<li>"$0"</li>"; next
```

```
    }
    /^[[:space:]]*$/ { flush(); next }

    { escape(); if (para!="") para=para" "; para=para$0 }

    END {
        flush()
        if (in_table) print "</pre>"
        if (in_code)  print "</code></pre>"
    }
    ' "$1"
}

# Inline markdown â HTML.  Runs after DONATELLO_html_body, so
# content is already HTML-escaped; only markdown punctuation remains.
DONATELLO_inline_sed() {
    sed         -e 's/``]*`/<code>1<code>/g'         -e 's/*]*strong>1<strong>/g'
}

# groff -ms block structure.  Mirrors DONATELLO_html_body but emits
# ms(7) macros.  Inline markup is applied by the ms_inline() awk
# function using match()/RSTART/RLENGTH loops.
DONATELLO_ms_body() {
    awk '
    BEGIN { in_code=0; in_table=0; para="" }

    function flush() {
        if (para!="") { print ".PP"; print para; para="" }
    }
    function ms_inline(s,   r,pre,mid,post,bra) {
        r=s
        while(match(r,/) {
            pre=substr(r,1,RSTART-1); mid=substr(r,RSTART+2,RLENGTH-4)
            post=substr(r,RSTART+RLENGTH); r=pre"\fB"mid"\fR"post }
        while(match(r,/+)) {
            pre=substr(r,1,RSTART-1); mid=substr(r,RSTART+1,RLENGTH-2)
            post=substr(r,RSTART+RLENGTH); r=pre"\fI"mid"\fR"post }
        while(match(r,/]+])]+/)) {
            pre=substr(r,1,RSTART-1)
            bra=index(substr(r,RSTART),"]"); mid=substr(r,RSTART+1,bra-2)
            post=substr(r,RSTART+RLENGTH); r=pre mid post }
        while(match(r,/`[^`]+`/)) {
            pre=substr(r,1,RSTART-1); mid=substr(r,RSTART+1,RLENGTH-2)
            post=substr(r,RSTART+RLENGTH); r=pre"\f(CW"mid"\fR"post }
        return r
    }

    /^```/ {
        if(!in_code){flush();print".DS L0ft CR";in_code=1}
        else{print".ft0DE";in_code=0}; next }
    in_code { print; next }

    /^/ {
        if(/^[-: |]+[[:space:]]*$/) next
```

```
        if(!in_table){flush();print".DS L0ft CR";in_table=1}
        print; next }
    in_table&&!/^/ { print".ft0DE"; in_table=0 }

    /^#### /{flush();sub(/^#### /,"");print".IP
    /^### / {flush();sub(/^### /,""); print".SS "ms_inline($0);next}
    /^## /  {flush();sub(/^## /,"");  print".SH "ms_inline($0);next}
    /^# /   {flush();sub(/^# /,"");   print".SH "ms_inline($0);next}

    /^---[[:space:]]*$/ { flush(); print".br0sp 0.5v"; next }

    /^> / {
        flush(); sub(/^> /,"")
        print".RS0I0ms_inline($0)"0RE"; next }
    /^[-*] / {
        flush(); sub(/^[-*] /,"")
        print".IP \(bu 20ms_inline($0); next }

    /^[[:space:]]*$/ { flush(); next }
    { line=ms_inline($0); if(para!="")para=para" "; para=para line }

    END {
        flush()
        if(in_code)  print".ft0DE"
        if(in_table) print".ft0DE"
    }
    ' "$1"
}


# ãã Shared filesystem scanner (BEBOP + ROCKSTEADY) ãããããããããããããããããããããããããããã
# Emits TSV: CAT  DATE  TITLE BASE  HAS_HTML    HAS_PDF
# Title is line 2 of the .txt file (MICHELANGELO's fixed format).
# PDF is considered present if it exists in either ~/html/ or the gopher dir.
_scan_gopher_dir() {
    _sg_dir="$1" _sg_cat="$2"
    for _f in "$_sg_dir"/[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]-*.txt; do
        [ -f "$_f" ] || continue
        _base="${_f##*/}"; _base="${_base%.txt}"
        _date=$(printf '%s' "$_base" | cut -c1-10)
        _title=$(sed -n '2p' "$_f" 2>/dev/null) || true
        [ -n "$_title" ] || _title="$_base"
        if [ -n "$_sg_cat" ]; then
            _html_dir="${HOME}/html/${_sg_cat}"
        else
            _html_dir="${HOME}/html"
        fi
        _hh=0; [ -f "${_html_dir}/${_base}.html" ] && _hh=1
        _hp=0
        { [ -f "${_html_dir}/${_base}.pdf" ] ||           [ -f "${_sg_dir}/${_base}.pdf" ]
        printf '%s      %s    %s    %s    %s    %s0             "$_sg_cat" "$_date" "$_tit
    done
}
```

```
# ãã Returns 0 (true) if a category dir contains at least one dated post ãããããã
_cat_has_posts() {
    for _chp in "$1"/[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]-*.txt; do
        [ -f "$_chp" ] && return 0
    done
    return 1
}

# ãã BUILD_NAV: generate nav bar from ~/gopher/ category subfolders ãããããããããã
BUILD_NAV() {
    printf '<nav>0a href="%s/index.html">index</a>' "$HTTP_BASE"
    for _d in "${HOME}/gopher"/*/; do
        [ -d "$_d" ] || continue
        _cat_has_posts "$_d" || continue
        _cn="${_d%/}"; _cn="${_cn##*/}"
        printf '  |  <a href="%s/index.html#%s">%s</a>' "$HTTP_BASE" "$_cn" "$_c
    done
    printf '0/nav>0
}

# ====================================================================
# LEONARDO  â  Stage 3: HTML Assembly + Template Injection
# "We work as a team.  I assemble the final output."
# ====================================================================
LEONARDO() {
    APRIL_say "LEONARDO: Generate HTML"

    BODY="$TMP/html_body.html"
    DONATELLO_html_body "$INPUT" | DONATELLO_inline_sed > "$BODY"

    # Extract reusable blocks from the template with awk range patterns,
    # then patch the placeholder strings with sed.
    T_STYLE=$(awk '/<style>/,/<style>/{print}' "$TEMPLATE")

    T_HEADER=$(awk '/<header>/,/<header>/{print}' "$TEMPLATE"          | sed             -e

    T_FOOTER=$(awk '/<footer>/,/<footer>/{print}' "$TEMPLATE"          | sed             -e

    {
        printf '<!DOCTYPE html>0html lang="en">0head>0
        printf '  <meta charset="UTF-8">0
        printf '  <meta name="viewport" content="width=device-width, initial-scale=1.0">0
        printf '  <title>%s</title>0 "$TITLE_RAW"
        printf '%s0 "$T_STYLE"
        printf '</head>0body>0
        printf '%s0 "$T_HEADER"
        BUILD_NAV
        printf '<article>0
        cat "$BODY"
        printf '0/article>0
        printf '%s0 "$T_FOOTER"
        printf '<p>Lynx compatible.</p>0/body>0/html>0
    } > "$HTML_OUT"
```

```
    APRIL_note "Written: $HTML_OUT"
}


# ======================================================================
# MICHELANGELO  â  Stage 4: Gopher ASCII Conversion
# "Gopher. Duuude.  It's like the web but *honest*."
# ======================================================================
MICHELANGELO() {
    APRIL_say "MICHELANGELO: Generate Gopher text"

    BODY="$TMP/gopher_body.txt"

    # Single sed pass: strip all markdown markers.
    # Code fence lines become blank lines (content is preserved).
    # Heading levels map to proportional indentation.
    sed         -e 's/^```.*$//'         -e 's/^#### /    /'         -e 's/^### /   /'

    {
        APRIL_rule
        printf '%s0 "$TITLE_RAW"
        APRIL_rule
        printf 'Date:   %s0uthor: %s0 "$DATE" "$AUTHOR"
        APRIL_rule
        echo ""
        cat "$BODY"
        echo ""
        APRIL_rule
        printf 'Published: %s0 "$(date)"
        APRIL_rule
    } > "$TXT_OUT"

    APRIL_note "Written: $TXT_OUT"
}


# ======================================================================
# FUGITOID (Professor Honeycutt)  â  Stage 5: PDF via groff -ms
#   ENEMY: KRANG
# "My robotic chassis is optimally suited for operating a groff pipeline.
#  Also I have defeated Krang before.  Several times."
# ======================================================================
FUGITOID() {
    APRIL_say "FUGITOID vs KRANG: Generate PDF"

    MS="$TMP/body.ms"

    ABSTRACT=$(awk         '/^## Abstract/{f=1;next} f&&/^## /{exit} f{print}'         "$I

    {
        printf '.ND0
        printf '.TL7s0 "$TITLE_RAW"
        printf '.AU7s0 "$AUTHOR"
        if [ -n "$ABSTRACT" ]; then
```

```
            printf '.AB7s0AE0 "$ABSTRACT"
        fi
        DONATELLO_ms_body "$INPUT"
    } > "$MS"

    if command -v ps2pdf >/dev/null 2>&1         && groff -ms -Tps "$MS" 2>/dev/null | ps2
        APRIL_note "Written: $PDF_GOPHER"
    elif groff -ms -Tpdf "$MS" > "$PDF_GOPHER" 2>/dev/null         && [ -s "$PDF_GOPHER" ]
        APRIL_note "Written: $PDF_GOPHER (native PDF)"
    else
        APRIL_warn "KRANG: groff pipeline failed â android body retreating; preserving .ms
        cp "$MS" "${HOME}/gopher/${BASE}.ms"
        APRIL_note "ms source: ~/gopher/${BASE}.ms"
    fi

    if [ -s "$PDF_GOPHER" ]; then
        cp "$PDF_GOPHER" "$PDF_HTML"
        APRIL_note "Copied:  $PDF_HTML"
    fi
}


# ======================================================================
# CASEY JONES  â  Stage 6: Gophermap Regeneration
#   ENEMY: BEBOP
# "I don't know what a gophermap is but I will absolutely hit it
#  with this hockey stick until it works."
#
# Gophermap format (tab-delimited):
#   <type><display>     <selector>  <host>      <port>
#   i  = info line (no link)
#   1  = gopher directory
#   h  = HTML link (selector = URL:http://...)
#   0  = text file
#   9  = binary (PDF)
#
# Fully regenerated from filesystem every run â no anchor markers.
# Category directory links at top; entries grouped newest-first.
# ======================================================================
CASEY_JONES() {
    APRIL_say "CASEY JONES vs BEBOP: Regenerate gophermap"

    ENTRIES="$TMP/casey_entries.tsv"
    : > "$ENTRIES"
    _scan_gopher_dir "${HOME}/gopher" "" >> "$ENTRIES"
    for _cdir in "${HOME}/gopher"/*/; do
        [ -d "$_cdir" ] || continue
        _cn="${_cdir%/}"; _cn="${_cn##*/}"
        _scan_gopher_dir "$_cdir" "$_cn"
    done >> "$ENTRIES"

    sort -t "$(printf ' ')" -k1,1 -k2,2r "$ENTRIES"         > "$TMP/casey_sorted.tsv" 2>/d

    mkdir -p "$(dirname "$GOPHERMAP")"
```

```
    {
        printf 'i fake  fake  00
        printf 'i  %s @ sdf.org     fake  fake  00 "$USER"
        printf 'i fake  fake  00

        # Category directory links (only for non-empty categories)
        for _cdir in "${HOME}/gopher"/*/; do
            [ -d "$_cdir" ] || continue
            _cat_has_posts "$_cdir" || continue
            _cn="${_cdir%/}"; _cn="${_cn##*/}"
            printf '1  %s/    %s/%s %s    %s0                "$_cn" "$GOPHER_BASE" "$_cn"
        done
        printf 'i fake  fake  00

        # Entry list grouped by category, newest first
        awk -F'   '             -v gbase="$GOPHER_BASE"             -v goph="$GOPHER_HOST"
        {
            cat=$1; date=$2; title=$3; base=$4; hh=$5; hp=$6
            if (cat != cur_cat) {
                lbl = (cat=="") ? "General" : cat
                printf "i     fake  fake  00
                printf "i  == %s == fake  fake  00, lbl
                printf "i     fake  fake  00
                cur_cat = cat
            }
            gpath_txt = (cat!="") ? gbase"/"cat"/"base".txt" : gbase"/"base".txt"
            gpath_pdf = (cat!="") ? gbase"/"cat"/"base".pdf" : gbase"/"base".pdf"
            printf "i[%s] %s  fake  fake  00, date, title
            printf "0  txt:  %s.txt %s    %s    %s0, base, gpath_txt, goph, gport
            if (hp=="1")
                printf "9  pdf:  %s.pdf   %s    %s    %s0, base, gpath_pdf, goph, gport
            printf "i   fake  fake  00
        }
        ' "$TMP/casey_sorted.tsv"
    } > "$GOPHERMAP"

    chmod 644 "$GOPHERMAP"
    APRIL_note "BEBOP: gophermap seized â Bebop's warthog credentials revoked."
    APRIL_note "Written: $GOPHERMAP"
    APRIL_note "Entries: $(wc -l < "$ENTRIES" | tr -d ' ') posts"
}


# ================================================================
# MONDO GECKO  â  Stage 7: HTML Index Regeneration
#   ENEMY: ROCKSTEADY
# "Dude, I can outskate a rhino with my eyes closed.
#  Regenerating a sitemap?  Way easier."
#
# Scans ~/gopher/ for date-prefixed .txt files, generates
# ~/html/index.html grouped by category with mirror links.
# ================================================================
MONDO_GECKO() {
    APRIL_say "MONDO GECKO vs ROCKSTEADY: Regenerate index.html"
```

```
SITEMAP="${HOME}/html/index.html"
ENTRIES="$TMP/fs_entries.tsv"
: > "$ENTRIES"

_scan_gopher_dir "${HOME}/gopher" "" >> "$ENTRIES"
for _cdir in "${HOME}/gopher"/*/; do
    [ -d "$_cdir" ] || continue
    _cn="${_cdir%/}"; _cn="${_cn##*/}"
    _scan_gopher_dir "$_cdir" "$_cn"
done >> "$ENTRIES"

# Sort: category ASC, date DESC (newest first within each section)
sort -t "$(printf ' ')" -k1,1 -k2,2r "$ENTRIES"         > "$TMP/entries_sorted.tsv" 2>

# ââ Template blocks ââââââââââââââââââââââââââââââââââââââââââââââââââââââ
T_STYLE=$(awk '/<style>/,/<style>/{print}' "$TEMPLATE")
T_HEADER=$(awk '/<header>/,/<header>/{print}' "$TEMPLATE"          | sed             -e
T_FOOTER=$(awk '/<footer>/,/<footer>/{print}' "$TEMPLATE"          | sed             -e

# ââ Generate category-grouped entry HTML ââââââââââââââââââââââââââââââââââ
# Structure (matches h1 > h2 > h3 > h4 hierarchy in the article):
#   <h3 id="cat">category</h3>
#   <h4><a href="...">Title</a> <small>[date]</small></h4>
#   <blockquote><small>gopher â pdf â mirror</small></blockquote>
# Title links to HTML when available, otherwise to gopher txt.
# blockquote gives one extra indent level beyond h4 in the template CSS.
ENTRIES_HTML="$TMP/entries.html"
awk -F' '          -v http="$HTTP_BASE"         -v goph="$GOPHER_HOST"         -v gport
{
    cat=$1; date=$2; title=$3; base=$4; hh=$5; hp=$6
    if (cat != cur_cat) {
        lbl = (cat=="") ? "general" : cat
        printf "<h3 id=
        cur_cat = cat
    }
    gpath = (cat!="") ? gbase"/"cat"/"base".txt" : gbase"/"base".txt"

    html_url   = (cat!="") ? http"/"cat"/"base".html"   : http"/"base".html"
    pdf_url    = (cat!="") ? http"/"cat"/"base".pdf"    : http"/"base".pdf"
    mirror_url = (mirror!="") ? ((cat!="") ? mirror"/"cat"/"base".html" : mirror"/"bas

    if (hh=="1")
        printf "<h4><a href=
    else
        printf "<h4>%s <small>[%s]</small></h4>0, title, date

    have_links = (hp=="1") || (mirror!="" && hh=="1")
    if (have_links) {
        printf "<blockquote><small>"
        sep = ""
        if (hp=="1") {
            printf "%s<a href=
            sep = " &mdash; "
        }
```

```
            if (mirror!="" && hh=="1")
                printf "%s<a href=
            printf "</small></blockquote>0
        }
    }
    ' "$TMP/entries_sorted.tsv" > "$ENTRIES_HTML"

    # ãã Assemble index.html ãããããããããããããããããããããããããããããããããããããããããããããããã
    {
        printf '<!DOCTYPE html>0html lang="en">0head>0
        printf '  <meta charset="UTF-8">0
        printf '  <meta name="viewport" content="width=device-width, initial-scale=1.0">0
        printf '  <title>%s</title>0 "$HTTP_BASE"
        printf '%s0 "$T_STYLE"
        printf '</head>0body>0
        printf '%s0 "$T_HEADER"
        BUILD_NAV
        printf '<article>0
        printf '<h1>Site Content</h1>0
        printf '<h2>File Listing</h2>0
        cat "$ENTRIES_HTML"
        printf '<hr>0
        printf '<p><a href="%s">%s</a></p>0 "$HTTP_BASE" "$HTTP_BASE"
        printf '<p>Personal site, writing, and notes.</p>0
        printf '<p>Content published under the <strong>MIT License</strong>. '
        printf 'This sitemap is auto-generated by '
        printf '<strong>shredder.sh</strong> and regenerated on each new post. '
        printf 'Titles link to the HTML version where available.</p>0
        printf '0/article>0
        printf '%s0 "$T_FOOTER"
        printf '<p>Lynx compatible.</p>0/body>0/html>0
    } > "$SITEMAP"

    APRIL_note "ROCKSTEADY: index.html seized â the rhino's grip on the sitemap is broken.
    APRIL_note "Written: $SITEMAP"
    APRIL_note "Scanned: $(wc -l < "$ENTRIES" | tr -d ' ') entries found"
}


# ====================================================================
# IRMA  â  Stage 8: File Permission Verification
#   ENEMY: BAXTER STOCKMAN
# "Baxter.  I work in a newsroom.  I've been making files world-readable
#  since before you had compound eyes.  Step aside."
#
# All published files must be 644 (rw-r--r--) and directories 755.
# Warns rather than aborts: the author may be working in a
# restrictive umask environment and should be informed, not blocked.
# ====================================================================
IRMA() {
    APRIL_say "IRMA vs BAXTER STOCKMAN: Verify permissions"
    ALL_OK=1

    # ãã 755 every directory we create or write into ãããããããããããããããããããããããã
```

```
    for DIR in "${HOME}/html" "${HOME}/gopher"                "${HOME}/gopher"/*/ "${HOME}
        [ -d "$DIR" ] || continue
        chmod 755 "$DIR" && APRIL_note "755 $DIR"             || { APRIL_warn "BAXTER STOC
    done

    # ää 644 every file in those directories (depth 1 each) ääääääääääääääääääääääääääää
    for DIR in "${HOME}/html" "${HOME}/gopher"                "${HOME}/gopher"/*/ "${HOME}
        [ -d "$DIR" ] || continue
        for F in "$DIR"/*; do
            [ -f "$F" ] || continue
            chmod 644 "$F" && APRIL_note "644 $F"                 || { APRIL_warn "BAXTER
        done
    done

    if [ "$ALL_OK" -eq 1 ]; then
        APRIL_note "BAXTER STOCKMAN: compound eyes confiscated â permissions set."
    else
        APRIL_warn "BAXTER STOCKMAN: one or more chmod calls failed â review warnings abov
    fi
}


# ====================================================================
# BATTLE NARRATIONS  â  Splinter dispatches allies against villains.
# Each function picks one of four lines via _nidx so the speech
# varies between runs and between stages in the same run.
# Villains have no dialogue.  They only produce error messages.
# ====================================================================

_narrate_vs_krang() {
    case "$(_nidx)" in
        0) SPLINTER_narrate "Professor Honeycutt! Krang has seized the groff pipeline and
        1) SPLINTER_narrate "Fugitoid â you have defeated Krang's technology across three
        2) SPLINTER_narrate "Krang believes that science and typesetting belong to him alo
        3) SPLINTER_narrate "The android body stands at the typesetter. I have watched thi
    esac
}

_narrate_vs_bebop() {
    case "$(_nidx)" in
        0) SPLINTER_narrate "Casey Jones! Bebop guards the gophermap with the intellect of
        1) SPLINTER_narrate "Mr. Jones â you are undisciplined, reckless, and you never re
        2) SPLINTER_narrate "Casey, Bebop is standing on the gophermap server and claiming
        3) SPLINTER_narrate "The warthog mutant does not understand filesystem scanning. H
    esac
}

_narrate_vs_rocksteady() {
    case "$(_nidx)" in
        0) SPLINTER_narrate "Mondo Gecko! Rocksteady has planted himself in front of our i
        1) SPLINTER_narrate "Gecko â stop skating in circles and focus. Rocksteady believe
        2) SPLINTER_narrate "Rocksteady does not understand category grouping. He does not
        3) SPLINTER_narrate "I will not pretend Mondo Gecko was my first choice for sitema
    esac
```

```
}

_narrate_vs_baxter() {
    case "$(_nidx)" in
        0) SPLINTER_narrate "Irma! Baxter Stockman â in his fly form â is attempting to co
        1) SPLINTER_narrate "Baxter's compound eyes see everything except a properly set p
        2) SPLINTER_narrate "The fly scientist believes he can prevent public access to ou
        3) SPLINTER_narrate "Irma â Baxter Stockman stands at the permission gate, bzzing
    esac
}

# ====================================================================
# SHREDDER  â  Main Orchestrator
# Calls the Foot Clan in order.  Does not tolerate failure.
# ====================================================================
SHREDDER() {
    case "$(_nidx)" in
        0) SPLINTER_narrate "I shall begin with structural validation. If you cannot count
        1) SPLINTER_narrate "Before allies can fight, the scroll must be examined. Unbalan
        2) SPLINTER_narrate "Structure first. Always. A markdown file with mismatched fenc
        3) SPLINTER_narrate "I have validated ten thousand manuscripts in my years. Most w
    esac
    SPLINTER            # 1. validate structure

    case "$(_nidx)" in
        0) SPLINTER_narrate "Raphael. I know you find spell-checking beneath you. I find y
        1) SPLINTER_narrate "Raphael â a warrior who cannot spell is a warrior who cannot
        2) SPLINTER_narrate "The spell check begins. Raphael will object. Raphael always o
        3) SPLINTER_narrate "Raph, I need your red mask on this document â not to fight it
    esac
    RAPHAEL             # 2. spell check

    case "$(_nidx)" in
        0) SPLINTER_narrate "Leonardo. You are the leader of this pipeline. Assemble the H
        1) SPLINTER_narrate "Leo â the template exists for a reason. The CSS is set. Your
        2) SPLINTER_narrate "Leonardo, the HTML stage is yours. Every great structure is b
        3) SPLINTER_narrate "The blue mask leads. Leonardo assembles HTML. Headers, body,
    esac
    LEONARDO            # 3. generate HTML

    case "$(_nidx)" in
        0) SPLINTER_narrate "Michelangelo! The Gopher text awaits. 72 columns. No exceptio
        1) SPLINTER_narrate "Mikey, I need your text wrapped to 72 columns for the Gopher
        2) SPLINTER_narrate "Michelangelo â Gopher is not dead. You will prove this by gen
        3) SPLINTER_narrate "The orange mask takes the plain-text stage. Gopher was the we
    esac
    MICHELANGELO        # 4. generate Gopher text

    _narrate_vs_krang
    FUGITOID            # 5. generate PDF  (vs KRANG)

    _narrate_vs_bebop
    CASEY_JONES         # 6. regenerate gophermap  (vs BEBOP)
```

```
    _narrate_vs_rocksteady
    MONDO_GECKO         # 7. regenerate index.html  (vs ROCKSTEADY)

    _narrate_vs_baxter
    IRMA                # 8. verify permissions  (vs BAXTER STOCKMAN)

    echo ""
    case "$(_nidx)" in
        0) SPLINTER_narrate "The pipeline has run its course. HTML, Gopher text, and PDF h
        1) SPLINTER_narrate "Krang retreats. Bebop is confused. Rocksteady is sitting down
        2) SPLINTER_narrate "Four enemies. Four allies. One published post. This is what t
        3) SPLINTER_narrate "I have seen many pipelines fail at this exact point. This one
    esac
    APRIL_say "SHREDDER: Mission complete."
    APRIL_note "HTML    : $HTML_OUT"
    APRIL_note "Text    : $TXT_OUT"
    APRIL_note "PDF     : $PDF_GOPHER"
    APRIL_note "PDF/web : $PDF_HTML"
    APRIL_note "Gopher  : $GOPHERMAP"
    APRIL_note "Sitemap : ${HOME}/html/index.html"
    echo ""
    printf '    Gopher: %s0 "$GOPHER_LINK"
    printf '    Web:    %s0 "$SITEMAP_URL"
}

SHREDDER
```