# GPT Patcher

ChatGPT can provide a unified patch as output for code related changes (project instructions or prompt example 'return diffs of the changes proposed for ease of applying them.'). These unified patches are in a format not understood by git or the patch command. They do not contain specific line numbers but context surrounding the changes. To avoid issues with GPT hallucinations, strict checks need to be performed.

The script below can help apply these patches. It accepts file, stdin, clipboard or interactive manual paste as source for the patch, parses it for the target filename, checks if it can apply the patch and outputs the newly patched file. Command line option -i can replace the original file immediately, but creates backup files. Command line option -t can be used to patch a different filename. On errors, the script will abort.

gpt-patcher.py

```
#!/usr/bin/env python3
import argparse
import os
import re
import sys
import shutil

try:
    import pyperclip
except ImportError:
    pyperclip = None


PATCH_BEGIN = "*** Begin Patch"
PATCH_END = "*** End Patch"
PATCH_UPDATE_FILE = "*** Update File:"


def read_patch_input(source: str = None):
    """Read patch from file, stdin, clipboard, or prompt."""
    if source:
        with open(source, "r", encoding="utf-8") as f:
            data = f.read()
    elif not sys.stdin.isatty():
        data = sys.stdin.read()
    else:
        clip = pyperclip.paste().strip() if pyperclip else ""
        if clip and PATCH_BEGIN in clip and PATCH_END in clip:
            print("Using patch from clipboard.")
            data = clip
        else:
            if pyperclip:
                print("No valid patch detected in clipboard.")
```

```
            print("Please paste your patch below, then press Ctrl-D
(Linux/macOS) or Ctrl-Z (Windows) when done:\n")
            try:
                data = sys.stdin.read()
            except KeyboardInterrupt:
                sys.exit("\nAborted.")
    return data.strip()


def validate_patch_format(data: str):
    """Ensure patch contains Begin/End Patch and Update File
headers."""
    if PATCH_BEGIN not in data or PATCH_END not in data:
        sys.exit("Error: Patch must contain '*** Begin Patch' and '***
End Patch'.")
    if PATCH_UPDATE_FILE not in data:
        sys.exit("Error: Patch must contain '*** Update File:' line.")
    return True


def parse_patch(data: str):
    """Extract the target filename and list of hunks."""
    match = re.search(r"\*\*\* Update File:\s*(.+)", data)
    if not match:
        sys.exit("Error: Could not find file in patch header.")
    filename = match.group(1).strip()

    hunks = []
    current_hunk = []
    in_hunk = False

    for line in data.splitlines():
        if line.startswith("@@"):
            if current_hunk:
                hunks.append(current_hunk)
            current_hunk = []
            in_hunk = True
        elif line.strip() == PATCH_END:
            if current_hunk:
                hunks.append(current_hunk)
            break
        elif in_hunk:
            current_hunk.append(line)

    return filename, hunks


def make_backup(path: str):
    """Create sequential .bak backups if needed."""
    base_backup = path + ".bak"
    if not os.path.exists(base_backup):
```

```
            shutil.copy2(path, base_backup)
            return base_backup

        i = 1
        while True:
            backup_name = f"{base_backup}{i}"
            if not os.path.exists(backup_name):
                shutil.copy2(path, backup_name)
                return backup_name
            i += 1


def find_and_replace_block(content, old_block, new_block,
context_before=None, context_after=None, filename=None):
    """Find a full old_block with optional context and replace with
new_block."""
    pattern_parts = []
    if context_before:
        pattern_parts.append(re.escape(context_before))
    pattern_parts.append(re.escape(old_block))
    if context_after:
        pattern_parts.append(re.escape(context_after))

    pattern = "(?s)" + ".*?".join(pattern_parts)
    match = re.search(pattern, content)
    if not match:
        # try without requiring both contexts (looser match)
        match = re.search(re.escape(old_block), content)
        if not match:
            sys.exit(
                f"Error: Could not find patch target block in
'{filename}' containing:\n{old_block.strip()[:200]}"
            )

    start, end = match.span()
    return content[:start] + content[start:end].replace(old_block,
new_block, 1) + content[end:]


def apply_hunk_to_content(content: str, hunk, filename):
    """Apply a hunk (multi-line context aware)."""
    minus_lines, plus_lines, context_lines = [], [], []

    for line in hunk:
        if line.startswith("-"):
            minus_lines.append(line[1:])
        elif line.startswith("+"):
            plus_lines.append(line[1:])
        elif line.startswith(" "):
            context_lines.append(line[1:])
```

```
    old_block = "\n".join(minus_lines)
    new_block = "\n".join(plus_lines)

    # extract leading and trailing context if available
    context_before = "\n".join(context_lines[:3]) if context_lines else
None
    context_after = "\n".join(context_lines[-3:]) if context_lines else
None

    return find_and_replace_block(content, old_block, new_block,
context_before, context_after, filename)


def apply_patch_to_file(filename, hunks):
    """Apply patch hunks to file contents."""
    with open(filename, "r", encoding="utf-8") as f:
        content = f.read()

    for hunk in hunks:
        content = apply_hunk_to_content(content, hunk, filename)

    return content


def main():
    parser = argparse.ArgumentParser(description="Apply ChatGPT unified
diffs to files.")
    parser.add_argument("patchfile", nargs="?", help="File containing
the patch (optional)")
    parser.add_argument("-i", "--in-place", action="store_true",
help="Replace original file (create backup)")
    parser.add_argument("-t", "--target", help="Apply patch to this
file instead of the one mentioned in patch")

    args = parser.parse_args()

    data = read_patch_input(args.patchfile)
    validate_patch_format(data)
    patch_file, hunks = parse_patch(data)

    target_file = args.target or patch_file

    if not os.path.exists(target_file):
        sys.exit(f"Error: Target file '{target_file}' not found.")

    result = apply_patch_to_file(target_file, hunks)

    if args.in_place:
        backup = make_backup(target_file)
        with open(target_file, "w", encoding="utf-8") as f:
            f.write(result)
```

```
            print(f"Patched {target_file} (backup: {backup})")
        else:
            sys.stdout.write(result)


if __name__ == "__main__":
    main()
```