

Session Transcript

Scripting SFTP and Red Hat Customer Portal uploads with curl in POSIX sh

Transcript generated October 25, 2026 (session dated July 2, 2026).

Redactions: the Red Hat refresh token and the SFTP token that appeared in the original conversation have been replaced with [REDACTED] placeholders. All pasted script listings are included in full so the edits can be followed version by version.

You

In a shell script, what is the easiest way to pass parameters such as password and put commands to sftp?

Claude

Short answer: **use SSH key-based authentication instead of a password**, and pass commands via a batch file or heredoc with `sftp -b`. That's the standard, secure, script-friendly pattern. If you're stuck with password auth, `sshpass` is the common (though less secure) workaround.

1. Best practice: SSH keys + batch file (no password needed)

```
#!/bin/bash
sftp -i /path/to/private_key -b /path/to/commands.txt user@host
```

commands.txt:

```
cd /remote/dir
put localfile.txt
bye
```

Or inline with a heredoc instead of a separate file:

```
#!/bin/bash
sftp -i /path/to/private_key user@host <<EOF
cd /remote/dir
put localfile.txt
bye
EOF
```

Set up key auth once (`ssh-keygen`, then copy the public key to the server's `~/.ssh/authorized_keys`), and you never need to handle a password in the script at all, which is more secure and avoids storing credentials in plaintext.

2. If you must use a password

`sftp` has no built-in flag for passing a password, so people commonly use `sshpass`:

```
#!/bin/bash
sshpass -p 'yourpassword' sftp -oBatchMode=no user@host <<EOF
cd /remote/dir
put localfile.txt
bye
EOF
```

Caveats:

- `sshpass` puts the password in the process list unless you use `-f passwordfile` or `-e` (env variable `SSHPASS`) instead of `-p`.

- Storing plaintext passwords in scripts is a real security risk. File permissions matter (chmod 600), and it's easy to accidentally commit them to version control.

```
export SSHPASS='yourpassword'
sshpass -e sftp user@host <<EOF
put localfile.txt
bye
EOF
```

3. Alternative: lftp

If you want more flexibility (globbing, mirroring, retries), lftp supports password and commands more cleanly than raw sftp:

```
lftp -u user,password sftp://host <<EOF
cd /remote/dir
put localfile.txt
bye
EOF
```

Recommendation: go with option 1 (keys + batch/heredoc) for anything beyond a one-off manual test. It avoids the whole password-handling problem entirely.

You

This application needs a password.

Claude

Got it. If key-based auth isn't an option, here's the cleanest way to handle it with a password.

Using sshpass (most common approach)

Install it first if needed: `sudo apt install sshpass` (Debian/Ubuntu) or `sudo yum install sshpass` (RHEL/CentOS).

Safest variant: password in an environment variable:

```
#!/bin/bash
export SSHPASS='yourpassword'

sshpass -e sftp -oBatchMode=no user@host <<EOF
cd /remote/dir
put localfile.txt
bye
EOF
```

Or password in a protected file:

```
#!/bin/bash
sshpass -f /path/to/passfile sftp -oBatchMode=no user@host <<EOF
cd /remote/dir
put localfile.txt
bye
EOF
```

```
chmod 600 /path/to/passfile # restrict access to owner only
```

Avoid `sshpass -p 'yourpassword'` directly. The password becomes visible to anyone running `ps aux` while the script executes, since it's passed as a command-line argument.

A few practical notes

- **Batch mode:** `-oBatchMode=no` is needed because `BatchMode=yes` disables password prompts entirely, expecting key auth.
- **Host key checking:** the first time you connect to a new host, SSH may prompt to confirm the host key, which will hang a script. Pre-accept it manually once, or (less secure) add `-oStrictHostKeyChecking=no`.
- **Permissions on the script/passfile:** since a password is stored somewhere on disk, lock down file permissions (`chmod 600`) and restrict who can read the script itself.
- **Better long-term fix:** if this is a system you control, see if password auth can be supplemented with key auth even for this one app/account. If it's a third-party system that only offers passwords, `sshpass` with a password file is the standard approach.

Want help wrapping this into a fuller script with error handling (e.g., checking the transfer succeeded, retry logic)?

You

Here is the problem I want to solve. As you can see, a tool generates a temporary password.

```
bearer_token=$(curl -s --request POST \
  'https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token' \
  --data-urlencode 'client_id=rhsm-api' \
  --data-urlencode 'grant_type=refresh_token' \
  --data-urlencode 'refresh_token=[REDACTED - refresh token]' \
  | jq -r '.access_token')
# Generate the SFTP token
curl -s --request POST \
  'https://access.redhat.com/hydra/rest/v2/sftp/token/' \
  -H "Authorization: Bearer $bearer_token" \
  -H 'Content-Type: application/json' \
  -d '{"expiryInDays": 90}'
# Now use sftp
sftp rhn-support-gscott@sftp.access.redhat.com
```

Claude

Nice, this is a clean use case. You just need to capture the token from that second `curl` response, then feed it to `sftp` as the password with `sshpass`. Here's the full pattern:

```
#!/bin/bash
set -euo pipefail

# Get bearer token
bearer_token=$(curl -s --request POST \
  'https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token' \
  --data-urlencode 'client_id=rhsm-api' \
  --data-urlencode 'grant_type=refresh_token' \
  --data-urlencode 'refresh_token=YOUR_REFRESH_TOKEN' \
  | jq -r '.access_token')

# Generate the SFTP token and capture it
sftp_token=$(curl -s --request POST \
  'https://access.redhat.com/hydra/rest/v2/sftp/token/' \
  -H "Authorization: Bearer $bearer_token" \
  -H 'Content-Type: application/json' \
  -d '{"expiryInDays": 90}' \
  | jq -r '.token') # <-- adjust key name to match actual JSON response

# Use the token as the SFTP password
```

```
sshpass -p "$sftp_token" sftp -oBatchMode=no rhn-support-gscott@sftp.access.redhat.com <<EOF
pwd
ls
bye
EOF
```

A few things to check/adjust:

1. **JSON field name:** I used `.token` as a placeholder. Run the second `curl` manually once and pipe it to `jq .` to see the actual field name, then update the `jq -r '.xxx'` accordingly.
2. **Token reuse note:** since you're requesting `expiryInDays: 90`, this token is valid for 90 days. If you're running this script repeatedly, you probably don't want to generate a brand-new token every single run. Consider either generating it once and storing it securely (a `chmod 600` file with its expiry date), or only regenerating it when needed.
3. **Avoid -p exposure:** like before, `sshpass -p` puts the token in the process list momentarily. Instead you could do:

```
export SSHPASS="$sftp_token"
sshpass -e sftp -oBatchMode=no rhn-support-gscott@sftp.access.redhat.com <<EOF
put /path/to/file
bye
EOF
unset SSHPASS
```

4. **Redact your refresh token:** the one in your message is a live credential pasted in plaintext. You may want to rotate/revoke it if this was a real token and not a placeholder.

Want me to help fill in the actual `put/cd` commands for what you're trying to upload?

You

But I cannot count on systems having `sshpass` installed.

Claude

Good constraint to flag. `sshpass` isn't always available and you can't assume it. The best fix here: **skip the `sftp` command entirely and use `curl`'s built-in SFTP support**, since `curl` is available almost everywhere and natively supports SFTP with username/password.

Upload with `curl` (recommended)

```
#!/bin/bash
set -euo pipefail

# ... (token retrieval as before) ...

curl -s --user "rhn-support-gscott:${sftp_token}" \
  -T /path/to/localfile.txt \
  sftp://sftp.access.redhat.com/remote/dir/
```

- `--user` takes username:password directly: no password prompt, no `sshpass`, no expect.
- `-T` uploads a file (like `put`).
- To download instead, just use `curl -o localfile sftp://.../remote/file` (no `-T`).
- To list a directory: `curl --user "user:pass" sftp://sftp.access.redhat.com/remote/dir/` (no `-T`, `curl` prints the directory listing).
- Multiple uploads/commands: run multiple `curl` invocations, or use `-Q` to send raw SFTP quote commands (e.g. `-Q "rename old new"`).

This avoids the whole "is sshpass/expect installed" problem, since `curl` is close to universally present, and it's a single static binary call rather than an interactive session.

If you truly need an interactive-style batch session

If you need multiple sequential SFTP operations in one connection and want to stick with `sftp` itself, the fallback without `sshpass` is `expect`:

```
#!/bin/bash
expect <<EOF
spawn sftp rhn-support-gscott@sftp.access.redhat.com
expect "password:"
send "${sftp_token}\r"
expect "sftp>"
send "put /path/to/localfile.txt\r"
expect "sftp>"
send "bye\r"
expect eof
EOF
```

But `expect` also isn't guaranteed to be installed, so this doesn't really solve your portability concern any better than `sshpass` does.

My recommendation: go with the `curl sftp://` approach. It's the most portable option since you're already using `curl` in this script anyway for the token calls, so there is no new dependency at all.

Want me to rewrite your full script end-to-end using this approach, including error handling if the token retrieval or upload fails?

You

I like the `curl sftp://` approach. How do I `cd` to the correct directory from inside `sftp`?

Claude

With `curl sftp://`, there's no interactive session to "`cd`" into. Each `curl` call is a one-shot connection. Instead, you just include the target directory directly in the URL path.

Absolute path

```
curl -s --user "rhn-support-gscott:${sftp_token}" \
-T /path/to/localfile.txt \
sftp://sftp.access.redhat.com/remote/absolute/dir/
```

Relative path (relative to the user's home/default directory)

```
curl -s --user "rhn-support-gscott:${sftp_token}" \
-T /path/to/localfile.txt \
sftp://sftp.access.redhat.com/relative/dir/
```

(Note: `curl`'s URL parsing means a single leading `/` right after the host is actually already "relative to home" for SFTP in most implementations, and `//` at the start signals an absolute path. This varies slightly by `curl` version, so it's worth testing once against the actual server to confirm which behavior you're getting.)

To check where you land / what's there

You can list a directory without uploading, just don't pass `-T`:

```
curl -s --user "rhn-support-gscott:${sftp_token}" \
```

```
sftp://sftp.access.redhat.com/
```

Run that once to see your default/home directory contents, and confirm the exact path Red Hat expects.

No native "cd", but you can chain quote commands

If you need to create a directory first (like `mkdir` before `put`), use `-Q`:

```
curl -s --user "rhn-support-gscott:${sftp_token}" \
-Q "mkdir /remote/newdir" \
-T /path/to/localfile.txt \
sftp://sftp.access.redhat.com/remote/newdir/
```

Supported `-Q` (quote) commands for SFTP in `curl` are limited to: `chgrp`, `chmod`, `chown`, `ln`, `mkdir`, `pwd`, `rename`, `rm`, `rmdir`, `symlink`. There is no `cd`, since navigation is just baked into the URL path per request.

Practical tip: run a quick discovery `curl` (no `-T`, just the bare URL) against Red Hat's SFTP root first to confirm the exact directory structure they expect uploads in.

You

This looks promising.

```
[gregs@storage2015 ~]$ bearer_token=$(curl -s --request POST \
'https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token' \
--data-urlencode 'client_id=rhsm-api' \
--data-urlencode 'grant_type=refresh_token' \
--data-urlencode 'refresh_token=[REDACTED - refresh token]' \
| jq -r '.access_token')
[gregs@storage2015 ~]$
[gregs@storage2015 ~]$ curl -s --request POST 'https://access.redhat.com/hydra/rest/v2/sftp/token/'
-H "Authorization: Bearer $bearer_token" -H 'Content-Type: application/json' -d '{"expiryInDays":
90}'
{
  "username" : "rhn-support-gscott",
  "token" : "[REDACTED - sftp token]",
  "expiryDate" : "2026-09-30T16:36:56.144Z"
}[gregs@storage2015 ~]$
[gregs@storage2015 ~]$ curl -s --user "rhn-support-gscott:${sftp_token}" \
> -T ./greptest.txt \
> sftp://sftp.access.redhat.com/rhn-support-gscott/
[gregs@storage2015 ~]$
```

Claude

Good progress. The token retrieval worked. But there's a bug in the last command, which is why it silently did nothing useful.

The bug

```
curl -s --user "rhn-support-gscott:${<token value>}" \
```

`${<token value>}` is being treated as a **variable name**, not the literal string. Since you never set a variable with that name, this expands to an empty string, so you actually ran:

```
curl -s --user "rhn-support-gscott:" -T ./greptest.txt sftp://...
```

...which authenticated with a blank password. It probably failed silently because `-s` suppresses all error/progress output.

Fix

Capture the token into a shell variable from the JSON response, then reference that variable. Don't paste the token value into \${...}:

```
sftp_token=$(curl -s --request POST \
  'https://access.redhat.com/hydra/rest/v2/sftp/token/' \
  -H "Authorization: Bearer $bearer_token" \
  -H 'Content-Type: application/json' \
  -d '{"expiryInDays": 90}' \
  | jq -r '.token')

curl --user "rhnsupport-gscott:$sftp_token" \
  -T ./greptest.txt \
  sftp://sftp.access.redhat.com/rhnsupport-gscott/
```

I also dropped `-s` for now so you can actually see what's happening. Add it back once this is confirmed working.

One more check

Verify the upload actually happened by listing the directory afterward:

```
curl --user "rhnsupport-gscott:$sftp_token" \
  sftp://sftp.access.redhat.com/rhnsupport-gscott/
```

That should show `greptest.txt` in the listing if the upload succeeded.

One important flag: the refresh token you pasted earlier in this conversation is a **live, valid credential** (it generated a real bearer token and SFTP token just now). Since it's now sitting in this chat transcript, you should treat it as compromised and **revoke/rotate it** in the Red Hat SSO account settings once you're done testing, and avoid pasting the real value in the future (use a placeholder like `refresh_token=$REFRESH_TOKEN` instead).

You

woops

Claude

No worries, it happens. Quick next steps:

1. **Revoke the refresh token.** Log into your Red Hat account (likely under `access.redhat.com` or `sso.redhat.com` account/session management) and look for active sessions or API tokens tied to `rhsm-api`, then revoke/invalidate it.
2. **Generate a new refresh token** to replace it once the old one is revoked.
3. **Going forward**, keep secrets out of pasted text by using placeholders like:

```
--data-urlencode "refresh_token=$REFRESH_TOKEN"
```

and store the actual value in an environment variable or a `chmod 600` file that's sourced at runtime, e.g.:

```
source ~/.redhat_creds # sets REFRESH_TOKEN=...
```

The SFTP token is lower-stakes since it's scoped just to SFTP and expires, but the same idea applies. If you don't need it anymore, no harm in letting it sit, but for anything long-lived, rotating is the safe move.

Want help restructuring the script so the refresh token lives in a separate sourced file instead of inline?

You

How about this?

```
[gregs@storage2015 ~]$ curl -s --user "rhnsupport-gscott:[sftp token]" -T ./greptest.txt
sftp://sftp.access.redhat.com/rhnsupport-gscott/
[gregs@storage2015 ~]$
```

Claude

That looks right. The syntax bug is fixed. With `-s`, curl stays silent on success (no output is actually a good sign), so let's just confirm the upload landed:

```
curl --user "rhnsupport-gscott:[sftp token]" \
sftp://sftp.access.redhat.com/rhnsupport-gscott/
```

If `greptest.txt` shows up in that listing, you're done. The upload worked.

A couple of follow-ups worth doing now that the mechanics are proven out:

1. **Check the exit code** to catch failures programmatically instead of relying on visual inspection:

```
curl -s --user "rhnsupport-gscott:[sftp token]" -T ./greptest.txt sftp://sftp.access.redhat.com/rhnsupport-gscott/
if [ $? -eq 0 ]; then
    echo "Upload succeeded"
else
    echo "Upload failed"
fi
```

2. **Swap the hardcoded token back to the variable** now that you know the command syntax works:

```
curl -s --user "rhnsupport-gscott:${sftp_token}" -T ./greptest.txt sftp://sftp.access.redhat.com/rhnsupport-gscott/
```

3. Since the SFTP token is now also sitting in this chat transcript, it's a short-lived, narrowly-scoped token (not as sensitive as the refresh token), but if you want to be thorough you could regenerate it too.

Once you confirm the listing shows the file, you've got a fully working, portable, `sshpass`-free upload pipeline.

You

Looks like we have a good starting point.

```
[gregs@storage2015 ~]$ cp greptest.txt greptest-20260702.txt
[gregs@storage2015 ~]$ curl -s --user "rhnsupport-gscott:[sftp token]" -T ./greptest-20260702.txt
sftp://sftp.access.redhat.com/rhnsupport-gscott/
[gregs@storage2015 ~]$ curl --user "rhnsupport-gscott:[sftp token]" sftp://sftp.access.redhat.com/rhnsupport-gscott/
-rwxr--r--  1      -      -      32 Jul  2 16:45 greptest-20260702.txt
-rwxr--r--  1      -      -      32 Jul  2 16:43 greptest.txt
[gregs@storage2015 ~]$
```

Claude

Confirmed working end-to-end. Both files are there with correct timestamps. The whole pipeline (token, SFTP token, curl upload, verify) is solid.

Here's a cleaned-up version pulling it all together, with the token stored in a variable (not hardcoded), error checking, and the refresh token pulled from an environment variable instead of pasted inline:

```
#!/bin/bash
```

```

set -euo pipefail

# Expects REFRESH_TOKEN to be set in the environment beforehand,
# e.g. `source ~/.redhat_creds` where that file sets REFRESH_TOKEN=...

USERNAME="rhn-support-gscott"
LOCAL_FILE="./gregtest-$(date +%Y%m%d).txt"
REMOTE_DIR="sftp://sftp.access.redhat.com/${USERNAME}/"

# Get bearer token
bearer_token=$(curl -s --request POST \
  'https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token' \
  --data-urlencode 'client_id=rhsm-api' \
  --data-urlencode 'grant_type=refresh_token' \
  --data-urlencode "refresh_token=${REFRESH_TOKEN}" \
  | jq -r '.access_token')

if [ -z "$bearer_token" ] || [ "$bearer_token" == "null" ]; then
  echo "Failed to get bearer token" >&2
  exit 1
fi

# Get SFTP token
sftp_token=$(curl -s --request POST \
  'https://access.redhat.com/hydra/rest/v2/sftp/token/' \
  -H "Authorization: Bearer $bearer_token" \
  -H 'Content-Type: application/json' \
  -d '{"expiryInDays": 90}' \
  | jq -r '.token')

if [ -z "$sftp_token" ] || [ "$sftp_token" == "null" ]; then
  echo "Failed to get SFTP token" >&2
  exit 1
fi

# Upload
curl -s --user "${USERNAME}:${sftp_token}" -T "$LOCAL_FILE" "$REMOTE_DIR"

if [ $? -eq 0 ]; then
  echo "Upload succeeded: $LOCAL_FILE"
else
  echo "Upload failed: $LOCAL_FILE" >&2
  exit 1
fi

```

Things worth thinking about as next steps:

- **Token reuse:** right now this regenerates a new 90-day SFTP token on every run. If this script runs on a schedule, you may want to cache the token (e.g. in a `chmod 600` file with its expiry date) and only regenerate when it's expired.
- **Rotate that leaked refresh token** when you get a chance, per the earlier note.

Want help adding the token-caching logic, or wiring this into a cron job?

You

Can we do this with `sh` instead of `bash`?

Claude

Sure. A few small adjustments make this fully POSIX `sh`-compatible. The main `bash`-isms to fix:

- `set -o pipefail` isn't POSIX, so drop it (or check exit codes manually where it matters)
- `==` in `[ ]` is a `bash` extension, so use `=` instead
- Everything else (`$(...)`, `[ -z ]`, functions, variables) is already POSIX-safe

```
#!/bin/sh
set -eu

# Expects REFRESH_TOKEN to be set in the environment beforehand,
# e.g. `./.redhat_creds` where that file sets REFRESH_TOKEN=...

USERNAME="rhn-support-gscott"
LOCAL_FILE="./gregtest-$(date +%Y%m%d).txt"
REMOTE_DIR="sftp://sftp.access.redhat.com/${USERNAME}/"

# Get bearer token
bearer_token=$(curl -s --request POST \
  'https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token' \
  --data-urlencode 'client_id=rhsm-api' \
  --data-urlencode 'grant_type=refresh_token' \
  --data-urlencode "refresh_token=${REFRESH_TOKEN}" \
  | jq -r '.access_token')

if [ -z "$bearer_token" ] || [ "$bearer_token" = "null" ]; then
  echo "Failed to get bearer token" >&2
  exit 1
fi

# Get SFTP token
sftp_token=$(curl -s --request POST \
  'https://access.redhat.com/hydra/rest/v2/sftp/token/' \
  -H "Authorization: Bearer $bearer_token" \
  -H 'Content-Type: application/json' \
  -d '{"expiryInDays": 90}' \
  | jq -r '.token')

if [ -z "$sftp_token" ] || [ "$sftp_token" = "null" ]; then
  echo "Failed to get SFTP token" >&2
  exit 1
fi

# Upload
if curl -s --user "${USERNAME}:${sftp_token}" -T "$LOCAL_FILE" "$REMOTE_DIR"; then
  echo "Upload succeeded: $LOCAL_FILE"
else
  echo "Upload failed: $LOCAL_FILE" >&2
  exit 1
fi
```

Notes on the changes:

1. `set -eu` instead of `set -euo pipefail`. `pipefail` is a bash/ksh extension not in POSIX sh. Since none of your pipelines here rely on catching a failure in the middle of the pipe, this isn't a problem for this script.
2. `[ "$x" = "null" ]` instead of `==`. POSIX `test`/`[` only supports `=` for string comparison.
3. Everything else is standard POSIX and works identically under `dash`, `ash`, `busybox sh`, etc.

One thing to double check: whichever system will run this needs `curl` and `jq` available, as `sh` doesn't imply anything about what other tools exist.

You

what does `set -eu` do?

Claude

`set -eu` is actually two separate options combined:

`set -e`: **exit on error**

Makes the script **stop immediately** if any command exits with a non-zero status, instead of continuing on to the next line.

Without it:

```
false          # this fails
echo "still runs" # ...but the script keeps going anyway
```

With `set -e`:

```
set -e
false          # this fails
echo "still runs" # never executes; script exits here
```

This is useful in a script like yours because if, say, the `curl` call to get the bearer token fails outright (network down, DNS fails), you don't want to blindly continue on to try using an empty/garbage token in the next step.

Caveats worth knowing:

- It doesn't apply inside conditionals. A command that fails inside an `if`, `while`, or before `&&||` won't trigger an exit, since its failure is being explicitly checked.
- It can behave inconsistently across different shells in edge cases involving pipelines and command substitution, which is one more reason the manual error checks you already have are a good belt-and-suspenders approach.

`set -u`: error on unset variables

Makes the script **fail immediately** if you try to use a variable that was never set, instead of silently treating it as an empty string.

Without it:

```
echo "$UNDEFINED_VAR" # prints nothing, script continues
```

With `set -u`:

```
set -u
echo "$UNDEFINED_VAR" # error: UNDEFINED_VAR: parameter not set; script exits
```

This is directly relevant to the bug you hit earlier (the token value used as a variable name silently expanding to nothing). If `set -u` had been active in that shell session, referencing an unset variable would have thrown an error immediately instead of quietly sending a blank password.

Why combine them as `-eu`

Together, they turn a script from "keep going and hope for the best" into "stop the moment something's clearly wrong," which is exactly the behavior you want for something touching credentials and file transfers, where a silent partial failure is worse than a loud early exit.

You

in your script, `$REFRESH_TOKEN` (upper case) is that long string, right?

Claude

Yes, exactly. `$REFRESH_TOKEN` is meant to hold that long JWT-looking string you pasted earlier (the one starting `eyJhbGciOiJIUzUxMiIs...`). That's your actual Red Hat SSO refresh token.

The script doesn't set it. It expects it to already exist in the environment when the script runs. So you need to set it beforehand, something like:

```
export REFRESH_TOKEN='eyJhbGciOiJIUzUxMiIs... (the real long value)...'
```

Or better, keep it out of your shell history and any script files entirely by putting it in a separate file:

```
# ~/.redhat_creds (chmod 600 this file!)
REFRESH_TOKEN='eyJhbGciOiJIUzUxMiIs... (the real long value)...'
```

Then before running the script:

```
. ~/.redhat_creds
./upload_script.sh
```

That `.` (dot) command sources the file into your current shell, which sets `REFRESH_TOKEN` as a variable available to the script that follows.

A couple of practical notes:

- `chmod 600 ~/.redhat_creds` so only you can read it.
- Don't export it inside `.redhat_creds` unless you need child processes/subshells to see it too. If you do run the script as `./upload_script.sh` (a genuinely separate process), you *do* need `export REFRESH_TOKEN=...` so it's inherited.
- Since this is the token that was exposed earlier in our conversation, whatever value you put in that file should be the **new, rotated one**, not the original you pasted.

You

Can I make that filename a parameter? Something like

```
param="/home/gregs/sso-credentials.txt"
```

```
. $param
```

Claude

Yes, that works fine in POSIX sh. A couple of small refinements worth adding:

```
#!/bin/sh
set -eu

param="/home/gregs/sso-credentials.txt"
. "$param"
```

Notes:

1. **Quote it:** `. "$param"` instead of `. $param`. Without quotes, if the path ever contained a space or glob character, word-splitting could break it.
2. **set -e + missing file:** if the file doesn't exist, the `.` command fails and `set -e` will immediately exit the script with a clear error, rather than silently continuing with `REFRESH_TOKEN` unset.
3. **Making it a true parameter (command-line arg):** if you meant letting the *caller* choose the file at runtime, use a positional parameter instead:

```
#!/bin/sh
set -eu

param="${1:-/home/gregs/sso-credentials.txt}"
. "$param"
```

This lets you run either:

```
./upload_script.sh # uses the default path
./upload_script.sh /path/to/other-credentials.txt # overrides it
```

`${1:-default}` means "use \$1 if it was passed, otherwise fall back to default."

4. **File permissions reminder:** `chmod 600 /home/gregs/sso-credentials.txt` so it's not world-readable, since it contains a live credential.

You

bummer

```
[gregs@storage2015 ~]$ sh -v ./case-upload-sftp.sh
ml () { module ml "$@"
}
which () { ( alias;
eval ${which_declare} ) | /usr/bin/which --tty-only --read-alias --read-functions --show-tilde --show-
dot $@
}
module () { local _mlredir=1;
if [ -n "${MODULES_REDIRECT_OUTPUT+x}" ]; then
if [ "$MODULES_REDIRECT_OUTPUT" = '0' ]; then
_mlredir=0;
else
if [ "$MODULES_REDIRECT_OUTPUT" = '1' ]; then
_mlredir=1;
fi;
fi;
fi;
case " $@" in
*' --no-redirect '* )
_mlredir=0
;;
*' --redirect '* )
_mlredir=1
;;
esac;
if [ $_mlredir -eq 0 ]; then
_module_raw "$@";
else
_module_raw "$@" 2>&1;
fi
}
scl () { if [ "$1" = "load" -o "$1" = "unload" ]; then
eval "module $@";
else
/usr/bin/scl "$@";
fi
}
_module_raw () { eval "${/usr/bin/tclsh '/usr/share/Modules/libexec/modulecmd.tcl' bash "$@"}";
_mlstatus=$?;
return $_mlstatus
}
#!/bin/sh
set -eu
# Expects REFRESH_TOKEN to be set in the environment beforehand,
# e.g. `./redhat_creds` where that file sets REFRESH_TOKEN=...
TOKEN_FILE="./RedHatSSOtoken.txt"
USERNAME="rhn-support-gscott"
LOCAL_FILE="./gregtest-$(date +%Y%m%d).txt"
REMOTE_DIR="sftp://sftp.access.redhat.com/${USERNAME}/"
. "$TOKEN_FILE"
REFRESH_TOKEN="[REDACTED - refresh token, note the leading = before the token]"
# Get bearer token
bearer_token=$(curl -s --request POST \
'https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token' \
```

```

--data-urlencode 'client_id=rhsm-api' \
--data-urlencode 'grant_type=refresh_token' \
--data-urlencode "refresh_token=${REFRESH_TOKEN}" \
| jq -r '.access_token')
if [ -z "$bearer_token" ] || [ "$bearer_token" = "null" ]; then
    echo "Failed to get bearer token" >&2
    exit 1
fi
Failed to get bearer token
[gregs@storage2015 ~]$

```

Claude

Two things going on here:

1. Stray = in your token file

Look at the actual line that got sourced:

```
REFRESH_TOKEN="=eyJhbGciOiJIUzUxMiIs..."
```

There's a leading = **inside the quotes**, right after REFRESH_TOKEN=". That means the actual value of \$REFRESH_TOKEN starts with a literal = character, which makes it an invalid token. That's almost certainly why the SSO endpoint rejected it and access_token came back null.

This usually happens when a file was created like:

```
REFRESH_TOKEN=export REFRESH_TOKEN==eyJ...
```

or copy-paste duplicated the =. Check RedHatSSOtoken.txt and make sure the line reads exactly:

```
REFRESH_TOKEN=eyJhbGciOiJIUzUxMiIs...
```

One = between the variable name and the value, no leading =, no export needed unless a child process needs it.

2. This is the same original token. It needs to be rotated

The value in your token file is the exact same refresh token you pasted in this chat earlier. I flagged this before: since it appeared in this conversation, it should be treated as exposed and rotated in your Red Hat account, then the new value put in RedHatSSOtoken.txt.

Quick way to verify the fix before rerunning the whole script

```

. ./RedHatSSOtoken.txt
echo "$REFRESH_TOKEN" | head -c 5

```

That should print eyJhb with no leading = or other stray characters. Once that looks right (and ideally with a rotated token), rerun the script.

You

Looks better now.

```

[gregs@storage2015 ~]$ sh -v ./case-upload-sftp.sh
ml () { module ml "$@"
}
which () { ( alias;
    eval ${which_declare} ) | /usr/bin/which --tty-only --read-alias --read-functions --show-tilde --show-dot $@

```

```

}
module () { local _mlredir=1;
if [ -n "${MODULES_REDIRECT_OUTPUT+x}" ]; then
if [ "$MODULES_REDIRECT_OUTPUT" = '0' ]; then
_mlredir=0;
else
if [ "$MODULES_REDIRECT_OUTPUT" = '1' ]; then
_mlredir=1;
fi;
fi;
fi;
case " $@" in
*' --no-redirect '* )
_mlredir=0
;;
*' --redirect '* )
_mlredir=1
;;
esac;
if [ $_mlredir -eq 0 ]; then
_module_raw "$@";
else
_module_raw "$@" 2>&1;
fi
}
scl () { if [ "$1" = "load" -o "$1" = "unload" ]; then
eval "module $@";
else
/usr/bin/scl "$@";
fi
}
_module_raw () { eval "$(/usr/bin/tclsh '/usr/share/Modules/libexec/modulecmd.tcl' bash "$@")";
_mlstatus=$?;
return $_mlstatus
}
#!/bin/sh
set -eu
# Expects REFRESH_TOKEN to be set in the environment beforehand,
# e.g. `./.redhat_creds` where that file sets REFRESH_TOKEN=...
TOKEN_FILE="./RedHatSSOtoken.txt"
USERNAME="rhn-support-gscott"
LOCAL_FILE="./gregtest-$(date +%Y%m%d).txt"
REMOTE_DIR="sftp://sftp.access.redhat.com/${USERNAME}/"
. "$TOKEN_FILE"
REFRESH_TOKEN="[REDACTED - refresh token]"
# Get bearer token
bearer_token=$(curl -s --request POST \
  'https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token' \
  --data-urlencode 'client_id=rhsm-api' \
  --data-urlencode 'grant_type=refresh_token' \
  --data-urlencode "refresh_token=${REFRESH_TOKEN}" \
  | jq -r '.access_token')
if [ -z "$bearer_token" ] || [ "$bearer_token" = "null" ]; then
echo "Failed to get bearer token" >&2
exit 1
fi
# Get SFTP token
sftp_token=$(curl -s --request POST \
  'https://access.redhat.com/hydra/rest/v2/sftp/token/' \
  -H "Authorization: Bearer $bearer_token" \
  -H 'Content-Type: application/json' \
  -d '{"expiryInDays": 90}' \
  | jq -r '.token')
if [ -z "$sftp_token" ] || [ "$sftp_token" = "null" ]; then
echo "Failed to get SFTP token" >&2
exit 1
fi
# Upload
if curl -s --user "${USERNAME}:${sftp_token}" -T "$LOCAL_FILE" "$REMOTE_DIR"; then

```

```

    echo "Upload succeeded: $LOCAL_FILE"
else
    echo "Upload failed: $LOCAL_FILE" >&2
    exit 1
fi
Upload succeeded: ./greptest-20260702.txt
[gregs@storage2015 ~]$

```

Claude

That's a clean end-to-end run. Token retrieval, SFTP token generation, and upload all succeeded. Solid working script.

Two things worth flagging before you consider this done:

1. That refresh token is still exposed in this transcript

This is now the **third time** the same live refresh token has appeared here in full, since `sh -v` echoes the entire script including the sourced file contents. I'd strongly recommend rotating it in your Red Hat account now. Not because anything's necessarily gone wrong, but because a credential pasted repeatedly into a chat transcript shouldn't be trusted going forward.

2. `sh -v` prints your token file's contents every run

That's expected behavior. `-v` echoes each line before executing it, including the `. "$TOKEN_FILE"` source and the `REFRESH_TOKEN=...` line it pulls in. That's fine for debugging, but if this script ever runs unattended (cron, logged output, etc.), drop `-v` so the token doesn't end up sitting in a log file:

```
sh ./case-upload-sftp.sh
```

Once you've rotated the token and swapped the new value into `RedHatSS0token.txt`, you're in good shape to move this into production use, maybe as a cron job or wrapped with timestamp/retry logic depending on how it'll actually be used.

You

How I prompt for a variable?

If `[ -z $USERNAME ]` then (prompt for it).

Claude

In POSIX `sh`, use `read` to prompt, and fix a couple of syntax issues in what you wrote. `if/then` need to be on separate lines (or separated by `;`), `If` should be lowercase, and the variable should be quoted:

```

if [ -z "${USERNAME:-}" ]; then
    printf "Enter username: "
    read USERNAME
fi

```

A few details:

1. `"${USERNAME:-}"` instead of just `"$USERNAME"`. Since you have `set -u` active, referencing an unset variable would cause the script to exit with an error before you even get to check if it's empty. The `:-` gives it an empty-string fallback just for this check.
2. `printf` **instead of** `echo` for the prompt. `printf "Enter username: "` doesn't add a trailing newline, so the cursor stays right after the colon on the same line. `echo` behavior for trailing newlines varies slightly across shells.
3. `read USERNAME` reads a line of input from the terminal into that variable.

For a password (hide the input)

If you're prompting for something sensitive like a password, use `stty` to disable terminal echo:

```
if [ -z "${SFTP_PASSWORD:-}" ]; then
    printf "Enter password: "
    stty -echo
    read SFTP_PASSWORD
    stty echo
    printf "\n"
fi
```

`stty -echo` stops the terminal from displaying typed characters, `stty echo` restores normal display afterward, and the extra `printf "\n"` adds back the newline that didn't get echoed when you pressed Enter.

Applied to your script

```
if [ -z "${USERNAME:-}" ]; then
    printf "Enter username: "
    read USERNAME
fi
```

placed near the top, before `USERNAME` gets used later in the script, would let it fall back to an interactive prompt if it wasn't hardcoded or passed in some other way.

You

This blows up with "unbounded variable."

```
TOKEN_FILE="./RedHatSS0token.txt"
USERNAME="$1"
LOCAL_FILE="$2"
REMOTE_DIR="sftp://sftp.access.redhat.com/${USERNAME}/"
if [ -z "${USERNAME:-}" ]; then
    printf "Enter username: "
    read USERNAME
fi
if [ -z "${LOCAL_FILE:-}" ]; then
    printf "Enter the file you want to upload: "
    read LOCAL_FILE
fi
```

Claude

The problem is upstream of your `-z` checks. It's these two lines:

```
USERNAME="$1"
LOCAL_FILE="$2"
```

With `set -u` active, referencing `$1` or `$2` when the script was called with no arguments (or only one) is itself an "unbound variable" error. Positional parameters count too. The script dies right here, before it ever reaches your `-z` fallback logic.

Fix: use `${1:-}` / `${2:-}` instead

```
USERNAME="${1:-}"
LOCAL_FILE="${2:-}"
```

This gives \$1/\$2 an empty-string default if they weren't passed, which then lets your existing -z checks work as intended:

```
TOKEN_FILE="./RedHatSS0token.txt"
USERNAME="${1:-}"
LOCAL_FILE="${2:-}"

if [ -z "$USERNAME" ]; then
    printf "Enter username: "
    read USERNAME
fi

if [ -z "$LOCAL_FILE" ]; then
    printf "Enter the file you want to upload: "
    read LOCAL_FILE
fi

REMOTE_DIR="sftp://sftp.access.redhat.com/${USERNAME}/"
```

One other thing I moved: REMOTE_DIR **now comes after the prompts**, not before. In your original, REMOTE_DIR was built from \$USERNAME immediately after USERNAME="\$1", so if \$1 was empty and the prompt filled it in afterward, REMOTE_DIR would still have been built with the empty value. Building it after both prompts ensures it reflects the final value.

You

It still blows up with that unbounded variable error.

```
TOKEN_FILE="./RedHatSS0token.txt"
USERNAME="$1:-"
LOCAL_FILE="$2:-"
if [ -z "${USERNAME:-}" ]; then
    printf "Enter username: "
    read USERNAME
fi
if [ -z "${LOCAL_FILE:-}" ]; then
    printf "Enter the file you want to upload: "
    read LOCAL_FILE
fi
REMOTE_DIR="sftp://sftp.access.redhat.com/${USERNAME}/"
. "$TOKEN_FILE"
```

Claude

Close, but the syntax got mangled. You wrote:

```
USERNAME="$1:-"
```

That's \$1 (still triggers the unbound-variable error by itself) followed by the *literal characters* :- stuck onto the end of the string. The :- default syntax only works **inside curly braces** (\${1:-}), not as \$1:-. Without the braces, the shell doesn't recognize it as a default-value expression at all.

Fix: add the braces back

```
USERNAME="${1:-}"
LOCAL_FILE="${2:-}"
```

Full corrected block:

```
TOKEN_FILE="./RedHatSS0token.txt"
```

```

USERNAME="${1:-}"
LOCAL_FILE="${2:-}"

if [ -z "$USERNAME" ]; then
    printf "Enter username: "
    read USERNAME
fi

if [ -z "$LOCAL_FILE" ]; then
    printf "Enter the file you want to upload: "
    read LOCAL_FILE
fi

REMOTE_DIR="sftp://sftp.access.redhat.com/${USERNAME}/"
. "$TOKEN_FILE"

```

The pattern to remember: `$1:-` is meaningless shell syntax (just concatenation), while `${1:-}` is the actual "use `$1` if set, otherwise empty string" expansion. The braces are what activate the special `:-` behavior.

You

That's better.

```

[gregs@storage2015 ~]$ ./case-upload-sftp.sh
Enter username: rhn-support-gscott
Enter the file you want to upload: gregtest.txt
Uploading local file gregtest.txt to sftp://sftp.access.redhat.com/rhn-support-gscott/
Upload succeeded: gregtest.txt
[gregs@storage2015 ~]$

```

[The user also pasted the full text of case-upload-sftp.sh:]

```

#!/bin/sh
set -eu

# This script depends on an offline token that will populate a variable
# named REFRESH_TOKEN. To generate your Offline Token:
#   - Launch a web browser and go to `https://access.redhat.com/management/api`.
#   - Click "Generate Token."
#   - This gives a long-lived offline token that will never expire unless revoked.
#   - Copy the token, paste it into a txt file, and store a copy of it as
#     ./RedHatSSOtoken.txt. See below.
#   - This token represents you to the Red Hat Customer Portal.
#   - This is the only time you will need a browser.
# Parameters:
# $1 - your customer portal username.
# $2 - the file you want to upload.
#
# If you want your file attached to a case, the file name
# must follow a strict file format policy.
# CASEID_* or sosreport-CASEID-*.
# If the filename is valid, the file will be uploaded to the case
# as an attachment and deleted from the Red Hat Secure FTP.
# If the filename is invalid, it remains in the Red Hat Secure FTP
# bucket for 30 days, after which it is permanently deleted.
#
# Here are a few example filenames with valid and invalid formats
# for automatic attachment to support cases.
#   sosreport-hostname-4-12345678-2021-04-20-uhbkfag.tar.xz **(Valid)**
#       [default filename generated by `sosreport`. This will
#       attach the file to case number 12345678.]
#   02436811_sosreport-12387183.gz **(Valid)**
#   02436811_log-report.txt **(Valid)**
#   02436811-sosreport.gz *(Invalid)*
#   sosreport.gz *(Invalid)*
#   sosreport_02436811.gz *(Invalid)*
#   sosreport-02855523-testreport.gz **(Valid)**

```

```

#   sosreport_02855523-testreport.gz *(Invalid)*
#   sosrep_02855523-testreport.gz *(Invalid)*
#   02855523-testreport.gz *(Invalid)*
#   02855523_testreport.gz **(Valid)**
TOKEN_FILE="./RedHatSSOtoken.txt"
USERNAME="${1:-}"
LOCAL_FILE="${2:-}"
if [ -z "${USERNAME:-}" ]; then
    printf "Enter username: "
    read USERNAME
fi
if [ -z "${LOCAL_FILE:-}" ]; then
    printf "Enter the file you want to upload: "
    read LOCAL_FILE
fi
REMOTE_DIR="sftp://sftp.access.redhat.com/${USERNAME}/"
. "$TOKEN_FILE"
# Get bearer token
bearer_token=$(curl -s --request POST \
'https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token' \
--data-urlencode 'client_id=rhsm-api' \
--data-urlencode 'grant_type=refresh_token' \
--data-urlencode 'refresh_token=${REFRESH_TOKEN}' \
| jq -r '.access_token')
if [ -z "$bearer_token" ] || [ "$bearer_token" = "null" ]; then
    echo "Failed to get bearer token" >&2
    exit 1
fi
# Get SFTP token
sftp_token=$(curl -s --request POST \
'https://access.redhat.com/hydra/rest/v2/sftp/token/' \
-H 'Authorization: Bearer $bearer_token' \
-H 'Content-Type: application/json' \
-d '{"expiryInDays": 90}' \
| jq -r '.token')
if [ -z "$sftp_token" ] || [ "$sftp_token" = "null" ]; then
    echo "Failed to get SFTP token" >&2
    exit 1
fi
# Upload
echo "Uploading local file ${LOCAL_FILE} to ${REMOTE_DIR}"
if curl -s --user "${USERNAME}:${sftp_token}" -T "$LOCAL_FILE" "$REMOTE_DIR"; then
    echo "Upload succeeded: $LOCAL_FILE"
else
    echo "Upload failed: $LOCAL_FILE" >&2
    exit 1
fi

```

Claude

This is in good shape. Well documented, working end-to-end, and portable. Nice job pulling it together.

A few small things worth considering as polish, not blockers:

1. Filename validation

You've documented the naming rules in detail (must match CASEID_* or sosreport-CASEID-*), but the script doesn't actually enforce them. Since an invalid name just means the file sits around for 30 days instead of failing loudly, this is optional:

```

case "$(basename "$LOCAL_FILE")" in
  [0-9]*_*|sosreport-[0-9]*-*)
    ;;
  *)
    echo "Warning: filename doesn't match expected pattern (CASEID_* or sosreport-CASEID-*)." >&2
    echo "File will upload but won't auto-attach to a case." >&2

```

```
;;
esac
```

This just warns rather than blocking, since you may sometimes intentionally upload something not meant for a case.

2. Check the local file exists before uploading

```
if [ ! -f "$LOCAL_FILE" ]; then
    echo "Error: file '$LOCAL_FILE' not found." >&2
    exit 1
fi
```

Add this right after the LOCAL_FILE prompt block.

3. TOKEN_FILE existence check

```
if [ ! -f "$TOKEN_FILE" ]; then
    echo "Error: token file '$TOKEN_FILE' not found." >&2
    exit 1
fi
```

None of these are required. The script works as-is. They're just guardrails that save you a confusing error message six months from now.

One last reminder since it's been a recurring thread: that original refresh token from earlier in the conversation is still worth rotating if you haven't already.

You

This next script generates a syntax error on line 76.

[The user pasted the following (the paste was cut off partway through the bearer token check):]

```
#!/bin/sh
set -eu
# This script depends on an offline token that will populate a variable
# named REFRESH_TOKEN. To generate your Offline Token:
#   - Launch a web browser and go to `https://access.redhat.com/management/api`.
#   - Click "Generate Token."
#   - This gives a long-lived offline token that will never expire unless revoked.
#   - Copy the token, paste it into a txt file, and store a copy of it as
#     ./RedHatSS0token.txt. See below.
#   - This token represents you to the Red Hat Customer Portal.
#   - This is the only time you will need a browser.
# Parameters:
# $1 - your customer portal username.
# $2 - the file you want to upload.
# $3 - Support case number.
# $4 - Any comment you want to add to your case upload.
# Since this script uploads files directly to the Red Hat Customer Portal,
# there are no constraints on file names to upload.
TOKEN_FILE="./RedHatSS0token.txt"
USERNAME="${1:-}"
LOCAL_FILE="${2:-}"
CASE_NUMBER="${3:-}"
CASE_COMMENT="${4:-}"
if [ ! -f "$TOKEN_FILE" ]; then
    echo "Error: token file '$TOKEN_FILE' not found." >&2
    exit 1
fi
if [ -z "${USERNAME:-}" ]; then
    printf "Enter your Red Hat Customer Portal username: "
    read USERNAME
```

```

fi
if [ -z "${LOCAL_FILE:-}" ]; then
    printf "Enter the file you want to upload: "
    read LOCAL_FILE
fi
if [ ! -f "$LOCAL_FILE" ]; then
    echo "Error: file '$LOCAL_FILE' not found." >&2
    exit 1
fi
if [ -z "${CASE_NUMBER:-}" ]; then
    printf "Enter the customer portal case number: "
    read CASE_NUMBER
fi
if [ -z "${CASE_COMMENT:-}" ]; then
    printf "Enter any comment you want with your upload: "
    read CASE_COMMENT
fi
. "$TOKEN_FILE"
# Get bearer token
bearer_token=$(curl -s --request POST \
    'https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token' \
    --data-urlencode 'client_id=rhsm-api' \
    --data-urlencode 'grant_type=refresh_token' \
    --data-urlencode "refresh_token=${REFRESH_TOKEN}" \
    | jq -r '.access_token')
if [ -z "$bearer_token" ] || [ "$bearer_token" = "null" ]; then
    echo "Failed to get bearer token" >&2
    exit 1

```

Claude

The paste seems to have gotten cut off. It ends mid-block at line 58 (`exit 1` with no closing `fi` after it), so I can't see what's actually on line 76 or the code around it. Since this is a syntax error, the issue is likely a missing `fi`, an unmatched quote, or an unclosed `$(...)` somewhere between where this paste ends and line 76.

A couple of ways to track it down:

1. Use `sh -n` to check syntax without running it

```
sh -n ./case-upload-portal.sh
```

This parses the script and reports syntax errors without executing anything.

2. Common culprits for this exact class of error

- A missing `fi`. If one `if` block is missing its `fi`, the shell won't detect it until much further down, which is why the *reported* error line often doesn't match the *actual* missing line.
- An unclosed quote in a `curl -d` JSON payload.
- An unclosed `$(...)` command substitution.

3. Can you paste the rest of the script?

Could you share from the `exit 1` through line 76 (and beyond, if there's more)? Once I can see the actual code near that line, I can point to the exact issue rather than guessing.

You

Oh - sorry. I thought I grabbed all of it. Trying again.

[The user pasted the complete `case-upload-portal.sh`:]

```
#!/bin/sh
set -eu
```

```

# This script depends on an offline token that will populate a variable
# named REFRESH_TOKEN. To generate your Offline Token:
#   - Launch a web browser and go to `https://access.redhat.com/management/api`.
#   - Click "Generate Token."
#   - This gives a long-lived offline token that will never expire unless revoked.
#   - Copy the token, paste it into a txt file, and store a copy of it as
#     ./RedHatSS0token.txt. See below.
#   - This token represents you to the Red Hat Customer Portal.
#   - This is the only time you will need a browser.
# Parameters:
# $1 - your customer portal username.
# $2 - the file you want to upload.
# $3 - Support case number.
# $4 - Any comment you want to add to your case upload.
# Since this script uploads files directly to the Red Hat Customer Portal,
# there are no constraints on file names to upload.
TOKEN_FILE="./RedHatSS0token.txt"
USERNAME="${1:-}"
LOCAL_FILE="${2:-}"
CASE_NUMBER="${3:-}"
CASE_COMMENT="${4:-}"
if [ ! -f "$TOKEN_FILE" ]; then
    echo "Error: token file '$TOKEN_FILE' not found." >&2
    exit 1
fi
if [ -z "${USERNAME:-}" ]; then
    printf "Enter your Red Hat Customer Portal username: "
    read USERNAME
fi
if [ -z "${LOCAL_FILE:-}" ]; then
    printf "Enter the file you want to upload: "
    read LOCAL_FILE
fi
if [ ! -f "$LOCAL_FILE" ]; then
    echo "Error: file '$LOCAL_FILE' not found." >&2
    exit 1
fi
if [ -z "${CASE_NUMBER:-}" ]; then
    printf "Enter the customer portal case number: "
    read CASE_NUMBER
fi
if [ -z "${CASE_COMMENT:-}" ]; then
    printf "Enter any comment you want with your upload: "
    read CASE_COMMENT
fi
. "$TOKEN_FILE"
# Get bearer token
bearer_token=$(curl -s --request POST \
    'https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token' \
    --data-urlencode 'client_id=rhsm-api' \
    --data-urlencode 'grant_type=refresh_token' \
    --data-urlencode "refresh_token=${REFRESH_TOKEN}" \
    | jq -r '.access_token')
if [ -z "$bearer_token" ] || [ "$bearer_token" = "null" ]; then
    echo "Failed to get bearer token" >&2
    exit 1
fi
# Upload
echo "Uploading local file ${LOCAL_FILE} to case number ${CASE_NUMBER} with comment \"${CASE_COMMENT}\""
if [ curl -s -X POST \
    'https://access.redhat.com/hydra/rest/cases/${CASE_NUMBER}/attachments' \
    -H "Authorization: Bearer $bearer_token" \
    -F "file=@${LOCAL_FILE}" \
    -F "description=${CASE_COMMENT}" ]; then
    echo "Upload succeeded: $LOCAL_FILE"
else
    echo "Upload failed: $LOCAL_FILE" >&2
    exit 1
fi

```

Claude

Found it. The bug is here:

```
if [ curl -s -X POST \
  'https://access.redhat.com/hydra/rest/cases/${CASE_NUMBER}/attachments' \
  -H "Authorization: Bearer $bearer_token" \
  -F "file=@${LOCAL_FILE}" \
  -F "description=${CASE_COMMENT}" ]; then
```

[is the test command, not a generic "run this and check success" wrapper. It expects a single conditional expression, not an entire curl command with flags. The shell tries to parse the curl command as arguments to test, which is nonsense syntax and blows up.

Fix: drop the [] and put the command directly in the if

```
if curl -s -X POST \
  "https://access.redhat.com/hydra/rest/cases/${CASE_NUMBER}/attachments" \
  -H "Authorization: Bearer $bearer_token" \
  -F "file=@${LOCAL_FILE}" \
  -F "description=${CASE_COMMENT}"; then
  echo "Upload succeeded: $LOCAL_FILE"
else
  echo "Upload failed: $LOCAL_FILE" >&2
  exit 1
fi
```

if already runs any command and checks its exit status. No [] needed; [is only for comparisons/tests like [-f file] or [-z "\$var"].

One more bug I caught while fixing that line

```
'https://access.redhat.com/hydra/rest/cases/${CASE_NUMBER}/attachments'
```

This uses **single quotes**, which prevent variable expansion entirely. \${CASE_NUMBER} would be sent to the API literally instead of the actual case number. I changed it to **double quotes** in the fix above so the variable expands correctly.

Try that swap and rerun sh -n ./case-upload-portal.sh to confirm the syntax is now clean before doing a live test.

You

looks good.

```
[gregs@storage2015 ~]$ ./case-upload-portal.sh rhn-support-gscott gregtest.txt 04459124
Enter any comment you want with your upload: Test upload 2026-0702 1:26 p.m.
Uploading local file gregtest.txt to case number 04459124 with comment "Test upload 2026-0702 1:26 p.m."
[{"caseNumber": "04459124", "uuid": "56c72681-a30f-40f8-b875-82a6cacce864", "checksum": "79888449f76d2e93403377bd533d470daf3a9645073140775978eb43b8cbef36", "createdAt": "2026-07-02T18:26:26Z", "createdBy": "Scott, Greg", "description": "Test upload 2026-0702 1:26 p.m.", "fileName": "gregtest.txt", "fileType": "text/plain", "id": "a09Hn00005BBBlaIAH", "isArchived": false, "isDeprecated": false, "isPrivate": false, "lastModifiedDate": "2026-07-02T18:26:26Z", "link": "https://attachments.access.redhat.com/hydra/rest/cases/04459124/attachments/56c72681-a30f-40f8-b875-82a6cacce864", "modifiedBy": "Scott, Greg", "size": 32, "sizeKB": 0.03, "downloadRestricted": false}]Upload succeeded: gregtest.txt
[gregs@storage2015 ~]$
```

[The user then ran "cat case-upload-portal.sh" and pasted the corrected script:]

```

#!/bin/sh
set -eu
# This script depends on an offline token that will populate a variable
# named REFRESH_TOKEN. To generate your Offline Token:
#   - Launch a web browser and go to `https://access.redhat.com/management/api`.
#   - Click "Generate Token."
#   - This gives a long-lived offline token that will never expire unless revoked.
#   - Copy the token, paste it into a txt file, and store a copy of it as
#     ./RedHatSSOtoken.txt. See below.
#   - This token represents you to the Red Hat Customer Portal.
#   - This is the only time you will need a browser.
# Parameters:
# $1 - your customer portal username.
# $2 - the file you want to upload.
# $3 - Support case number.
# $4 - Any comment you want to add to your case upload.
# Since this script uploads files directly to the Red Hat Customer Portal,
# there are no constraints on file names to upload.
TOKEN_FILE="./RedHatSSOtoken.txt"
USERNAME="${1:-}"
LOCAL_FILE="${2:-}"
CASE_NUMBER="${3:-}"
CASE_COMMENT="${4:-}"
if [ ! -f "$TOKEN_FILE" ]; then
    echo "Error: token file '$TOKEN_FILE' not found." >&2
    exit 1
fi
if [ -z "${USERNAME:-}" ]; then
    printf "Enter your Red Hat Customer Portal username: "
    read USERNAME
fi
if [ -z "${LOCAL_FILE:-}" ]; then
    printf "Enter the file you want to upload: "
    read LOCAL_FILE
fi
if [ ! -f "$LOCAL_FILE" ]; then
    echo "Error: file '$LOCAL_FILE' not found." >&2
    exit 1
fi
if [ -z "${CASE_NUMBER:-}" ]; then
    printf "Enter the customer portal case number: "
    read CASE_NUMBER
fi
if [ -z "${CASE_COMMENT:-}" ]; then
    printf "Enter any comment you want with your upload: "
    read CASE_COMMENT
fi
. "$TOKEN_FILE"
# Get bearer token
bearer_token=$(curl -s --request POST \
    'https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token' \
    --data-urlencode 'client_id=rhsm-api' \
    --data-urlencode 'grant_type=refresh_token' \
    --data-urlencode "refresh_token=${REFRESH_TOKEN}" \
    | jq -r '.access_token')
if [ -z "$bearer_token" ] || [ "$bearer_token" = "null" ]; then
    echo "Failed to get bearer token" >&2
    exit 1
fi
# Upload
echo "Uploading local file ${LOCAL_FILE} to case number ${CASE_NUMBER} with comment \"${CASE_COMMENT}\""
if curl -s -X POST \
    "https://access.redhat.com/hydra/rest/cases/${CASE_NUMBER}/attachments" \
    -H "Authorization: Bearer $bearer_token" \
    -F "file=@${LOCAL_FILE}" \
    -F "description=${CASE_COMMENT}"; then
    echo "Upload succeeded: $LOCAL_FILE"
else

```

```

    echo "Upload failed: $LOCAL_FILE" >&2
    exit 1
fi

```

Claude

That worked perfectly. The file is now attached directly to case 04459124, confirmed by the JSON response with the case number, checksum, and a download link.

Both scripts are now solid: one for raw SFTP uploads, one for direct case-attachment uploads via the REST API. Nice, complete pair of tools.

A couple of small polish items if you want to take this further:

1. **Pretty-print the JSON response** instead of the raw blob, by piping through `jq` .:

```

response=$(curl -s -X POST \
  "https://access.redhat.com/hydra/rest/cases/${CASE_NUMBER}/attachments" \
  -H "Authorization: Bearer $bearer_token" \
  -F "file=@${LOCAL_FILE}" \
  -F "description=${CASE_COMMENT}")
echo "$response" | jq .

```

Note: capturing output into a variable like this means you lose the direct pass/fail from `if curl ...`; then, so you'd want to check `?` explicitly or inspect the JSON for an error field instead.

2. **HTTP status code check.** Right now, "success" is just "curl didn't have a connection-level failure," not "the API actually accepted the file." You could check the HTTP code explicitly:

```

http_code=$(curl -s -o /tmp/response.json -w "%{http_code}" -X POST \
  "https://access.redhat.com/hydra/rest/cases/${CASE_NUMBER}/attachments" \
  -H "Authorization: Bearer $bearer_token" \
  -F "file=@${LOCAL_FILE}" \
  -F "description=${CASE_COMMENT}")
if [ "$http_code" = "200" ] || [ "$http_code" = "201" ]; then
    echo "Upload succeeded: $LOCAL_FILE"
    cat /tmp/response.json
else
    echo "Upload failed (HTTP $http_code): $LOCAL_FILE" >&2
    cat /tmp/response.json >&2
    exit 1
fi

```

Neither is required. Your current version does the job.

You

How do we remove `/tmp/response.json` after displaying it?

Claude

Just add `rm -f /tmp/response.json` right after you're done reading it. `-f` avoids an error if the file somehow isn't there.

```

http_code=$(curl -s -o /tmp/response.json -w "%{http_code}" -X POST \
  "https://access.redhat.com/hydra/rest/cases/${CASE_NUMBER}/attachments" \
  -H "Authorization: Bearer $bearer_token" \
  -F "file=@${LOCAL_FILE}" \
  -F "description=${CASE_COMMENT}")

if [ "$http_code" = "200" ] || [ "$http_code" = "201" ]; then
    echo "Upload succeeded: $LOCAL_FILE"
    cat /tmp/response.json
    rm -f /tmp/response.json

```

```

else
    echo "Upload failed (HTTP $http_code): $LOCAL_FILE" >&2
    cat /tmp/response.json >&2
    rm -f /tmp/response.json
    exit 1
fi

```

A couple of refinements worth considering

1. Avoid duplicating the `rm` in both branches. Read the file into a variable once, delete it immediately, then use the variable:

```

response_body=$(cat /tmp/response.json)
rm -f /tmp/response.json

if [ "$http_code" = "200" ] || [ "$http_code" = "201" ]; then
    echo "Upload succeeded: $LOCAL_FILE"
    echo "$response_body"
else
    echo "Upload failed (HTTP $http_code): $LOCAL_FILE" >&2
    echo "$response_body" >&2
    exit 1
fi

```

2. Avoid a predictable filename in `/tmp`. Since `/tmp` is shared across users on most systems, a fixed name like `/tmp/response.json` is a minor risk (symlink attacks, race conditions, collisions if the script runs twice at once). Use `mktemp` instead:

```

tmpfile=$(mktemp)
http_code=$(curl -s -o "$tmpfile" -w "%{http_code}" -X POST \
    "https://access.redhat.com/hydra/rest/cases/${CASE_NUMBER}/attachments" \
    -H "Authorization: Bearer $bearer_token" \
    -F "file=@${LOCAL_FILE}" \
    -F "description=${CASE_COMMENT}")

response_body=$(cat "$tmpfile")
rm -f "$tmpfile"

```

`mktemp` creates a uniquely-named, securely-permissioned temp file and gives you its path.

You

[The user attached a terminal capture showing two test runs of `case-upload-portal.sh` and the script itself.]

```

[gregs@storage2015 ~]$ ./case-upload-portal.sh rhn-support-gscott gregtest.txt 12345678
Enter any comment you want with your upload: breaks
Uploading local file gregtest.txt to case number 12345678 with comment "breaks"
Upload failed (HTTP 400): gregtest.txt
{"message":"Failed to create attachment metadata","detailMessage":"Set is null or empty"}
[gregs@storage2015 ~]$ ./case-upload-portal.sh rhn-support-gscott gregtest.txt 04459124 "one more test
> "
Uploading local file gregtest.txt to case number 04459124 with comment "one more test
"
Upload succeeded: gregtest.txt
[{"caseNumber":"04459124","uuid":"da31249e-7383-4189-ac49-8c2edd429ece","checksum":"79888449f76d2e93403377bd533d470daf3a9645073140775978eb43b8cbef36","createdAt":"2026-07-02T18:40:29Z","createdBy":"Scott, Greg","description":"one more test","fileName":"gregtest.txt","fileType":"text/plain","id":"a09Hn00005BBBnvIAH","isArchived":false,"isDeprecated":false,"isPrivate":false,"lastModifiedDate":"2026-07-02T18:40:29Z","link":"https://attachments.access.redhat.com/hydra/rest/cases/04459124/attachments/da31249e-7383-4189-ac49-8c2edd429ece","modifiedBy":"Scott, Greg","size":32,"sizeKB":0.03,"downloadRestricted":false}]

```

Looks like we have a finished product.

[The attached script (the HTTP status code version):]

```
#!/bin/sh
set -eu

# This script depends on an offline token that will populate a variable
# named REFRESH_TOKEN. To generate your Offline Token:
#   - Launch a web browser and go to `https://access.redhat.com/management/api`.
#   - Click "Generate Token."
#   - This gives a long-lived offline token that will never expire unless revoked.
#   - Copy the token, paste it into a txt file, and store a copy of it as
#     ./RedHatSSOtoken.txt. See below.
#   - This token represents you to the Red Hat Customer Portal.
#   - This is the only time you will need a browser.

# Parameters:
# $1 - your customer portal username.
# $2 - the file you want to upload.
# $3 - Support case number.
# $4 - Any comment you want to add to your case upload.

# Since this script uploads files directly to the Red Hat Customer Portal,
# there are no constraints on file names to upload.

TOKEN_FILE="./RedHatSSOtoken.txt"
USERNAME="${1:-}"
LOCAL_FILE="${2:-}"
CASE_NUMBER="${3:-}"
CASE_COMMENT="${4:-}"

if [ ! -f "$TOKEN_FILE" ]; then
    echo "Error: token file '$TOKEN_FILE' not found." >&2
    exit 1
fi

if [ -z "${USERNAME:-}" ]; then
    printf "Enter your Red Hat Customer Portal username: "
    read USERNAME
fi

if [ -z "${LOCAL_FILE:-}" ]; then
    printf "Enter the file you want to upload: "
    read LOCAL_FILE
fi

if [ ! -f "$LOCAL_FILE" ]; then
    echo "Error: file '$LOCAL_FILE' not found." >&2
    exit 1
fi

if [ -z "${CASE_NUMBER:-}" ]; then
    printf "Enter the customer portal case number: "
    read CASE_NUMBER
fi

if [ -z "${CASE_COMMENT:-}" ]; then
    printf "Enter any comment you want with your upload: "
    read CASE_COMMENT
fi

. "$TOKEN_FILE"

# Get bearer token
bearer_token=$(curl -s --request POST \
    'https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token' \
    --data-urlencode 'client_id=rhsm-api' \
```

```

--data-urlencode 'grant_type=refresh_token' \
--data-urlencode "refresh_token=${REFRESH_TOKEN}" \
| jq -r '.access_token')

if [ -z "$bearer_token" ] || [ "$bearer_token" = "null" ]; then
    echo "Failed to get bearer token" >&2
    exit 1
fi

# Upload
echo "Uploading local file ${LOCAL_FILE} to case number ${CASE_NUMBER} with comment \"${CASE_COMMENT}\""
http_code=$(curl -s -o /tmp/response.json -w "%{http_code}" -X POST \
    "https://access.redhat.com/hydra/rest/cases/${CASE_NUMBER}/attachments" \
    -H "Authorization: Bearer $bearer_token" \
    -F "file=@${LOCAL_FILE}" \
    -F "description=${CASE_COMMENT}")
if [ "$http_code" = "200" ] || [ "$http_code" = "201" ]; then
    echo "Upload succeeded: $LOCAL_FILE"
    cat /tmp/response.json
    rm -f /tmp/response.json
else
    echo "Upload failed (HTTP $http_code): $LOCAL_FILE" >&2
    cat /tmp/response.json >&2
    rm -f /tmp/response.json
    exit 1
fi

```

Claude

Nice. This really is in good shape now. Both test runs actually confirm the error handling is working correctly:

- **Case 12345678:** that's not a real case number, so the API correctly rejected it with a 400 and a clear message, and your script caught that, printed the error, and exited non-zero instead of falsely claiming success.
- **Case 04459124** with a multi-line comment: worked fine, attachment created, JSON confirms it.

You've now got two solid, portable, sshpass-free scripts:

1. `case-upload-sftp.sh`: raw SFTP upload to the Red Hat SFTP bucket
2. `case-upload-portal.sh`: direct REST upload attached to a specific case, with real HTTP-status-based error handling

One tiny leftover from earlier: you're still using the fixed `/tmp/response.json` path rather than `mktemp`. Not a real problem for personal use, but if this script might ever run as multiple concurrent instances, swapping in `mktemp` avoids any chance of collisions. Purely optional.

Good, methodical work getting here. Each bug you hit (`${var}` vs `$var`: -, [] vs bare command, single vs double quotes, `set -u` with positional params) was a real, useful lesson. And nice job navigating the credential-exposure hiccups along the way. It's worth a final check that the refresh token got rotated once you're done testing.

You

That `TOKEN_FILE` should be one line formed like:

```
REFRESH_TOKEN="(Paste the long string from https://access.redhat.com/management/api)"
```

How would I validate it in the scripts?

Claude

Good idea. Right now a malformed token file just gets silently sourced and fails later at the bearer-token step with a less obvious error. Here's how to validate it explicitly, right after sourcing it.

1. Confirm `REFRESH_TOKEN` actually got set and isn't empty

```
. "$TOKEN_FILE"

if [ -z "${REFRESH_TOKEN:-}" ]; then
    echo "Error: REFRESH_TOKEN is empty or not set in '$TOKEN_FILE'." >&2
    echo "Expected format: REFRESH_TOKEN=\"(token from https://access.redhat.com/management/api)\"" >&2
    exit 1
fi
```

2. Sanity-check the token's shape (it's a JWT)

Offline/refresh tokens from Red Hat SSO are JWTs: three base64url segments separated by dots. You can catch obvious paste mistakes with a simple pattern check:

```
case "$REFRESH_TOKEN" in
    *.*.*)
        ;;
    *)
        echo "Error: REFRESH_TOKEN in '$TOKEN_FILE' doesn't look like a valid token (expected JWT format)."
        >&2
        exit 1
        ;;
esac
```

This is a loose check (just "does it contain at least two dots"), not real JWT validation, but it's enough to catch the common failure modes, like the stray = bug from earlier, or accidentally pasting only part of the token.

3. Put both checks together

```
if [ ! -f "$TOKEN_FILE" ]; then
    echo "Error: token file '$TOKEN_FILE' not found." >&2
    exit 1
fi

. "$TOKEN_FILE"

if [ -z "${REFRESH_TOKEN:-}" ]; then
    echo "Error: REFRESH_TOKEN is empty or not set in '$TOKEN_FILE'." >&2
    exit 1
fi

case "$REFRESH_TOKEN" in
    *.*.*)
        ;;
    *)
        echo "Error: REFRESH_TOKEN in '$TOKEN_FILE' doesn't look like a valid token." >&2
        exit 1
        ;;
esac
```

The real validation is the bearer-token exchange itself

Your existing check ([-z "\$bearer_token"] || ["\$bearer_token" = "null"]) is already the authoritative validation. A token can be perfectly well-formed and still be expired, revoked, or just wrong, and the only way to know for sure is asking Red Hat's SSO endpoint. The two checks above are just about catching *obvious* local mistakes early, with a clearer error message, before spending a network round-trip to find out.

You

What do you think about a `wc -l` test against the file to make sure it's one line?

Claude

That's a reasonable instinct, but it has a couple of sharp edges that make it less reliable than it looks.

The catch with `wc -l`

`wc -l` counts **newline characters**, not "lines of visible content." That creates two false-result scenarios:

1. **A single-line file with no trailing newline** (many editors and `printf` don't add one) reports 0, not 1. You'd reject a perfectly valid file.
2. **A file with a trailing blank line** or a stray comment line reports 2+ and you'd reject a file that's actually fine.

So `wc -l` is really testing "how many newlines are there," which only loosely correlates with "is this one clean `REFRESH_TOKEN=...` line."

A more robust alternative: count non-blank, non-comment lines

```
line_count=$(grep -vc '^[[:space:]]*$\|^[[:space:]]*#' "$TOKEN_FILE")

if [ "$line_count" -ne 1 ]; then
    echo "Error: '$TOKEN_FILE' should contain exactly one line (REFRESH_TOKEN=...)." >&2
    echo "Found $line_count non-blank/non-comment line(s)." >&2
    exit 1
fi
```

This strips out blank lines and comment lines before counting, so it tolerates a trailing newline, a stray blank line, or a `#` comment, while still catching the real problem case (someone pasted two tokens, or leftover debug lines).

But honestly, is this check pulling its weight?

Your two earlier checks (empty/unset `REFRESH_TOKEN`, and the JWT-shape case pattern) already catch the failure modes that actually happened to you in this conversation. A line-count check adds complexity to guard against a scenario (extra stray lines in the file) that hasn't actually come up.

If you want the safety net anyway, the `grep -vc` version above is the one I'd recommend over plain `wc -l`.

You

[The user pasted the full text of `case-upload-portal.sh` with the token validation moved to the top:]

```
#!/bin/sh
set -eu
# case-upload-portal.sh
# save this to a convenient location and then:
# cd /path/to/where/you/saved/this
# chmod 755 ./case-upload-portal.sh
# This script depends on an offline token that will populate a variable
# named REFRESH_TOKEN. To generate your Offline Token:
#   - Launch a web browser and go to `https://access.redhat.com/management/api`.
#   - Click "Generate Token."
#   - This gives a long-lived offline token that will never expire unless revoked.
#   - Copy the token, paste it into a txt file, and store a copy of it as
#     ./RedHatSSOtoken.txt. See below.
#   - This token represents you to the Red Hat Customer Portal.
#   - This is the only time you will need a browser.
# Parameters:
# $1 - your customer portal username.
# $2 - the file you want to upload.
# $3 - Support case number.
# $4 - Any comment you want to add to your case upload.
# Since this script uploads files directly to the Red Hat Customer Portal,
# there are no constraints on file names to upload.
TOKEN_FILE="./RedHatSSOtoken.txt"
# Modify this file path and name appropriately for your environment.
# The file needs one line. The format looks like:
```

```

# REFRESH_TOKEN="(Paste the long string from https://access.redhat.com/management/api)"
if [ ! -f "$TOKEN_FILE" ]; then
    echo "Error: token file '$TOKEN_FILE' not found." >&2
    exit 1
fi
. "$TOKEN_FILE"
if [ -z "${REFRESH_TOKEN:-}" ]; then
    echo "Error: REFRESH_TOKEN is empty or not set in '$TOKEN_FILE'." >&2
    exit 1
fi
case "$REFRESH_TOKEN" in
    *.*.*)
        ;;
    *)
        echo "Error: REFRESH_TOKEN in '$TOKEN_FILE' doesn't look like a valid token." >&2
        exit 1
        ;;
esac
USERNAME="${1:-}"
LOCAL_FILE="${2:-}"
CASE_NUMBER="${3:-}"
CASE_COMMENT="${4:-}"
if [ -z "${USERNAME:-}" ]; then
    printf "Enter your Red Hat Customer Portal username: "
    read USERNAME
fi
if [ -z "${LOCAL_FILE:-}" ]; then
    printf "Enter the file you want to upload: "
    read LOCAL_FILE
fi
if [ ! -f "$LOCAL_FILE" ]; then
    echo "Error: file '$LOCAL_FILE' not found." >&2
    exit 1
fi
if [ -z "${CASE_NUMBER:-}" ]; then
    printf "Enter the customer portal case number: "
    read CASE_NUMBER
fi
if [ -z "${CASE_COMMENT:-}" ]; then
    printf "Enter any comment you want with your upload: "
    read CASE_COMMENT
fi
# Get bearer token
bearer_token=$(curl -s --request POST \
    'https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token' \
    --data-urlencode 'client_id=rhsm-api' \
    --data-urlencode 'grant_type=refresh_token' \
    --data-urlencode "refresh_token=${REFRESH_TOKEN}" \
    | jq -r '.access_token')
if [ -z "$bearer_token" ] || [ "$bearer_token" = "null" ]; then
    echo "Failed to get bearer token" >&2
    exit 1
fi
# Upload
echo "Uploading local file ${LOCAL_FILE} to case number ${CASE_NUMBER} with comment \"${CASE_COMMENT}\""
http_code=$(curl -s -o /tmp/response.json -w "%{http_code}" -X POST \
    "https://access.redhat.com/hydra/rest/cases/${CASE_NUMBER}/attachments" \
    -H "Authorization: Bearer $bearer_token" \
    -F "file=@${LOCAL_FILE}" \
    -F "description=${CASE_COMMENT}")
if [ "$http_code" = "200" ] || [ "$http_code" = "201" ]; then
    echo "Upload succeeded: $LOCAL_FILE"
    cat /tmp/response.json
    rm -f /tmp/response.json
else
    echo "Upload failed (HTTP $http_code): $LOCAL_FILE" >&2
    cat /tmp/response.json >&2
    rm -f /tmp/response.json
    exit 1
fi

```

fi

Claude

This is a well-structured, complete script. Nice reorganization moving the token validation up front. The logical flow now is: validate token file, validate/collect user inputs, get bearer token, upload.

A few small things I noticed on this read-through, all optional:

1. Typo: "Custmer" should be "Customer"

```
printf "Enter your Red Hat Custmer Portal username: "
```

Minor, but worth a quick fix since it's user-facing.

2. The `chmod 755` comment

This makes the script world-readable and world-executable. Since the script itself doesn't contain secrets (those live in `RedHatSSOtoken.txt`), that's fine. Just flagging that if anyone ever inlines a token directly into this file instead of using the separate token file, 755 would expose it to every user on the box.

3. Still using the fixed `/tmp/response.json` path

This was the `mktemp` suggestion from earlier. Still optional.

4. One thing that's *not* a problem but worth confirming

`REFRESH_TOKEN` is never explicitly exported in `RedHatSSOtoken.txt`, and that's fine. Since you `. "$TOKEN_FILE"` (source it) rather than run it as a subprocess, the variable exists directly in the current shell's scope.

Everything else is solid

- Token file existence check
- Empty/unset `REFRESH_TOKEN` check
- Loose JWT-shape validation
- Positional params with `set -u`-safe defaults
- Interactive fallback prompts
- Local file existence check
- Bearer token failure check
- HTTP status code check (not just curl exit code)
- Temp file cleanup

This is genuinely production-ready for personal/team use. The only thing I'd still nudge you toward before calling it fully done: confirm that the original leaked refresh token has been rotated, since it's been in this transcript many times now.

You

[The user pasted the script again with the "Custmer" typo fixed:]

```
#!/bin/sh
set -eu
# case-upload-portal.sh
# save this to a convenient location and then:
# cd /path/to/where/you/saved/this
# chmod 755 ./case-upload-portal.sh
# This script depends on an offline token that will populate a variable
```

```

# named REFRESH_TOKEN. To generate your Offline Token:
#   - Launch a web browser and go to `https://access.redhat.com/management/api`.
#   - Click "Generate Token."
#   - This gives a long-lived offline token that will never expire unless revoked.
#   - Copy the token, paste it into a txt file, and store a copy of it as
#     ./RedHatSSOtoken.txt. See below.
#   - This token represents you to the Red Hat Customer Portal.
#   - This is the only time you will need a browser.
# Parameters:
# $1 - your customer portal username.
# $2 - the file you want to upload.
# $3 - Support case number.
# $4 - Any comment you want to add to your case upload.
# Since this script uploads files directly to the Red Hat Customer Portal,
# there are no constraints on file names to upload.
TOKEN_FILE="./RedHatSSOtoken.txt"
# Modify this file path and name appropriately for your environment.
# The file needs one line. The format looks like:
# REFRESH_TOKEN="(Paste the long string from https://access.redhat.com/management/api)"
if [ ! -f "$TOKEN_FILE" ]; then
    echo "Error: token file '$TOKEN_FILE' not found." >&2
    exit 1
fi
. "$TOKEN_FILE"
if [ -z "${REFRESH_TOKEN:-}" ]; then
    echo "Error: REFRESH_TOKEN is empty or not set in '$TOKEN_FILE'." >&2
    exit 1
fi
case "$REFRESH_TOKEN" in
    *.*.*)
        ;;
    *)
        echo "Error: REFRESH_TOKEN in '$TOKEN_FILE' doesn't look like a valid token." >&2
        exit 1
        ;;
esac
USERNAME="${1:-}"
LOCAL_FILE="${2:-}"
CASE_NUMBER="${3:-}"
CASE_COMMENT="${4:-}"
if [ -z "${USERNAME:-}" ]; then
    printf "Enter your Red Hat Customer Portal username: "
    read USERNAME
fi
if [ -z "${LOCAL_FILE:-}" ]; then
    printf "Enter the file you want to upload: "
    read LOCAL_FILE
fi
if [ ! -f "$LOCAL_FILE" ]; then
    echo "Error: file '$LOCAL_FILE' not found." >&2
    exit 1
fi
if [ -z "${CASE_NUMBER:-}" ]; then
    printf "Enter the customer portal case number: "
    read CASE_NUMBER
fi
if [ -z "${CASE_COMMENT:-}" ]; then
    printf "Enter any comment you want with your upload: "
    read CASE_COMMENT
fi
# Get bearer token
bearer_token=$(curl -s --request POST \
    'https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token' \
    --data-urlencode 'client_id=rhsm-api' \
    --data-urlencode 'grant_type=refresh_token' \
    --data-urlencode "refresh_token=${REFRESH_TOKEN}" \
    | jq -r '.access_token')
if [ -z "$bearer_token" ] || [ "$bearer_token" = "null" ]; then
    echo "Failed to get bearer token" >&2

```

```

    exit 1
fi
# Upload
echo "Uploading local file ${LOCAL_FILE} to case number ${CASE_NUMBER} with comment \"${CASE_COMMENT}\""
http_code=$(curl -s -o /tmp/response.json -w "%{http_code}" -X POST \
    "https://access.redhat.com/hydra/rest/cases/${CASE_NUMBER}/attachments" \
    -H "Authorization: Bearer $bearer_token" \
    -F "file=@${LOCAL_FILE}" \
    -F "description=${CASE_COMMENT}")
if [ "$http_code" = "200" ] || [ "$http_code" = "201" ]; then
    echo "Upload succeeded: $LOCAL_FILE"
    cat /tmp/response.json
    rm -f /tmp/response.json
else
    echo "Upload failed (HTTP $http_code): $LOCAL_FILE" >&2
    cat /tmp/response.json >&2
    rm -f /tmp/response.json
    exit 1
fi

```

Claude

This is essentially the final version. Typo fixed, everything else unchanged from the last read-through, and it's clean.

At this point the script covers all the important bases:

- Fails fast and clearly on a missing/malformed token file
- Validates required inputs, with sensible interactive fallback
- Confirms the local file actually exists before attempting upload
- Checks the real HTTP status code, not just curl's exit code
- Cleans up its temp file in both success and failure paths
- Well-commented for future you (or a teammate) picking this up cold

The only two items left on the table are both optional polish, not corrections:

1. mktemp instead of the fixed /tmp/response.json path. Matters only if this ever runs concurrently.
2. A comment noting *why* the token lives in a separate file rather than inline, given the chmod 755 on the script itself.

Neither blocks you from using this as-is. This is a solid, working pair of scripts (SFTP + direct case-attachment upload) that solve the original problem cleanly and portably, without depending on sshpass or expect.

You

See what you think of this explanation.

[The user pasted the final version of case-upload-portal.sh, reproduced below, with expanded header comments.]

```

#!/bin/sh
set -eu
# case-upload-portal.sh
# save this to a convenient location and then:
# cd /path/to/where/you/saved/this
# chmod 755 ./case-upload-portal.sh
# This script depends on an offline token that will populate a variable
# named REFRESH_TOKEN. To generate your Offline Token:
#   - Launch a web browser and go to `https://access.redhat.com/management/api`.
#   - Click "Generate Token."
#   - This gives a long-lived offline token that will never expire unless revoked.
#   - Copy the token, paste it into a txt file, and store a copy of it
#     someplace convenient. Keep this safe because it's your permanent
#     token to use the Red Hat Customer Portal via CLI. See below.
#   - This is the only time you will need a browser.

```

```

# Parameters:
# $1 - your customer portal username.
# $2 - the file you want to upload.
# $3 - Support case number.
# $4 - Any comment you want to add to your case upload.
# Since this script uploads files directly to the Red Hat Customer Portal,
# there are no constraints on file names to upload.
TOKEN_FILE="./RedHatSSOtoken.txt"
# Modify this file path and name appropriately for your environment.
# Make sure it's protected. For example, chmod 600 ./RedHatSSOtoken.txt
# The file needs one line. The format looks like:
# REFRESH_TOKEN="(Paste the long string from https://access.redhat.com/management/api)"
if [ ! -f "$TOKEN_FILE" ]; then
    echo "Error: token file '$TOKEN_FILE' not found." >&2
    exit 1
fi
. "$TOKEN_FILE"
if [ -z "${REFRESH_TOKEN:-}" ]; then
    echo "Error: REFRESH_TOKEN is empty or not set in '$TOKEN_FILE'." >&2
    exit 1
fi
case "$REFRESH_TOKEN" in
    *.*.*)
        ;;
    *)
        echo "Error: REFRESH_TOKEN in '$TOKEN_FILE' doesn't look like a valid token." >&2
        exit 1
        ;;
esac
USERNAME="${1:-}"
LOCAL_FILE="${2:-}"
CASE_NUMBER="${3:-}"
CASE_COMMENT="${4:-}"
if [ -z "${USERNAME:-}" ]; then
    printf "Enter your Red Hat Customer Portal username: "
    read USERNAME
fi
if [ -z "${LOCAL_FILE:-}" ]; then
    printf "Enter the file you want to upload: "
    read LOCAL_FILE
fi
if [ ! -f "$LOCAL_FILE" ]; then
    echo "Error: file '$LOCAL_FILE' not found." >&2
    exit 1
fi
if [ -z "${CASE_NUMBER:-}" ]; then
    printf "Enter the customer portal case number: "
    read CASE_NUMBER
fi
if [ -z "${CASE_COMMENT:-}" ]; then
    printf "Enter any comment you want with your upload: "
    read CASE_COMMENT
fi
# Get bearer token
bearer_token=$(curl -s --request POST \
    'https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token' \
    --data-urlencode 'client_id=rhsm-api' \
    --data-urlencode 'grant_type=refresh_token' \
    --data-urlencode "refresh_token=${REFRESH_TOKEN}" \
    | jq -r '.access_token')
if [ -z "$bearer_token" ] || [ "$bearer_token" = "null" ]; then
    echo "Failed to get bearer token" >&2
    exit 1
fi
# Upload
echo "Uploading local file ${LOCAL_FILE} to case number ${CASE_NUMBER} with comment \"${CASE_COMMENT}\""
http_code=$(curl -s -o /tmp/response.json -w "%{http_code}" -X POST \
    "https://access.redhat.com/hydra/rest/cases/${CASE_NUMBER}/attachments" \
    -H "Authorization: Bearer $bearer_token" \

```

```

        -F "file=@${LOCAL_FILE}" \
        -F "description=${CASE_COMMENT}")
if [ "$http_code" = "200" ] || [ "$http_code" = "201" ]; then
    echo "Upload succeeded: $LOCAL_FILE"
    cat /tmp/response.json
    rm -f /tmp/response.json
else
    echo "Upload failed (HTTP $http_code): $LOCAL_FILE" >&2
    cat /tmp/response.json >&2
    rm -f /tmp/response.json
    exit 1
fi

```

Claude

This is good. The added comments do exactly what good comments should do: explain the *why*, not just the *what*.

Specifically:

```

# Keep this safe because it's your permanent
# token to use the Red Hat Customer Portal via CLI. See below.

```

This correctly sets expectations that the offline token is long-lived and sensitive. Someone reading this for the first time now understands the stakes before they go pasting it somewhere careless (like, say, a chat window).

```

# Make sure it's protected. For example, chmod 600 ./RedHatSSOtoken.txt

```

This closes the gap I flagged earlier. It directly answers "why does the token live in a separate file instead of inline in a 755 script," right at the point where a reader would naturally wonder that.

Both additions are accurate and placed exactly where someone following the instructions would need them, not buried at the bottom, but right next to the relevant action (generating the token, creating the token file).

Nothing to correct here. This script and its documentation are genuinely done: functionally solid, properly validated, and now well-explained for anyone else who picks it up. Good finish.

You

Can you make a PDF transcript with all the dialog from this session?