DevOps Homework 1 - Winter 2026

Student: César Núñez
Date: 01/12/2026

Exercise 1: Git Basics and Configuration

Deliverables

- Screenshot of `git config --list --global` output:

```
(capp30255-advml) canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/my-notes$ git config --list --global
user.email=ca.nunezhuaman@gmail.com
user.name=Cesar Nunez
init.defaultbranch=main
```

- Screenshot of `git status` showing ignored files and `git log --oneline` output:

```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/my-notes$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        todo.txt

nothing added to commit but untracked files present (use "git add" to track)
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/my-notes$ git log --oneline
55748ee (HEAD -> main) Add more items to shopping list
aa9fe58 Initial commit with shopping list
```

Exercise 2: Git Branches

Deliverables

- Screenshot of final `notes.txt` file showing both work and personal notes merged together:

```
  GNU nano 7.2                                    notes.txt
Some general notes for DevOps on main branch
Adding work-related notes to the file on work-notes branch
Personal notes to the file on personal-notes branch
```

- Screenshot of `git log --graph --oneline` showing the branch history and merge commits:

```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/my-notes$ git log --graph --oneline
*   21339ad (HEAD -> main) Merge personal-notes to main
|\
| * 5a9a5ef Adding personal notes to notes.txt
* | 6d05969 Add work-related notes to notes.txt
|/
* afb5b77 Initializin repo with notes.txt
```

- Screenshot of conflict resolution process (if conflicts occurred during merge):

```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/my-notes$ git merge personal-notes
Auto-merging notes.txt
CONFLICT (content): Merge conflict in notes.txt
Automatic merge failed; fix conflicts and then commit the result.
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/my-notes$ nano notes.txt
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/my-notes$ git add .
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/my-notes$ git commit -m"Merge personal-notes to main"
[main 21339ad] Merge personal-notes to main
```

```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/my-notes$ git diff afb5b7795b8b22cf3f11b035fb5ed4d623f6fcaf 21339adddf134703ad00a81edebecb1d23ce5a51
diff --git a/notes.txt b/notes.txt
index e01a786..20c9a43 100644
--- a/notes.txt
+++ b/notes.txt
@@ -1 +1,3 @@
 Some general notes for DevOps on main branch
+Adding work-related notes to the file on work-notes branch
+Personal notes to the file on personal-notes branch
```

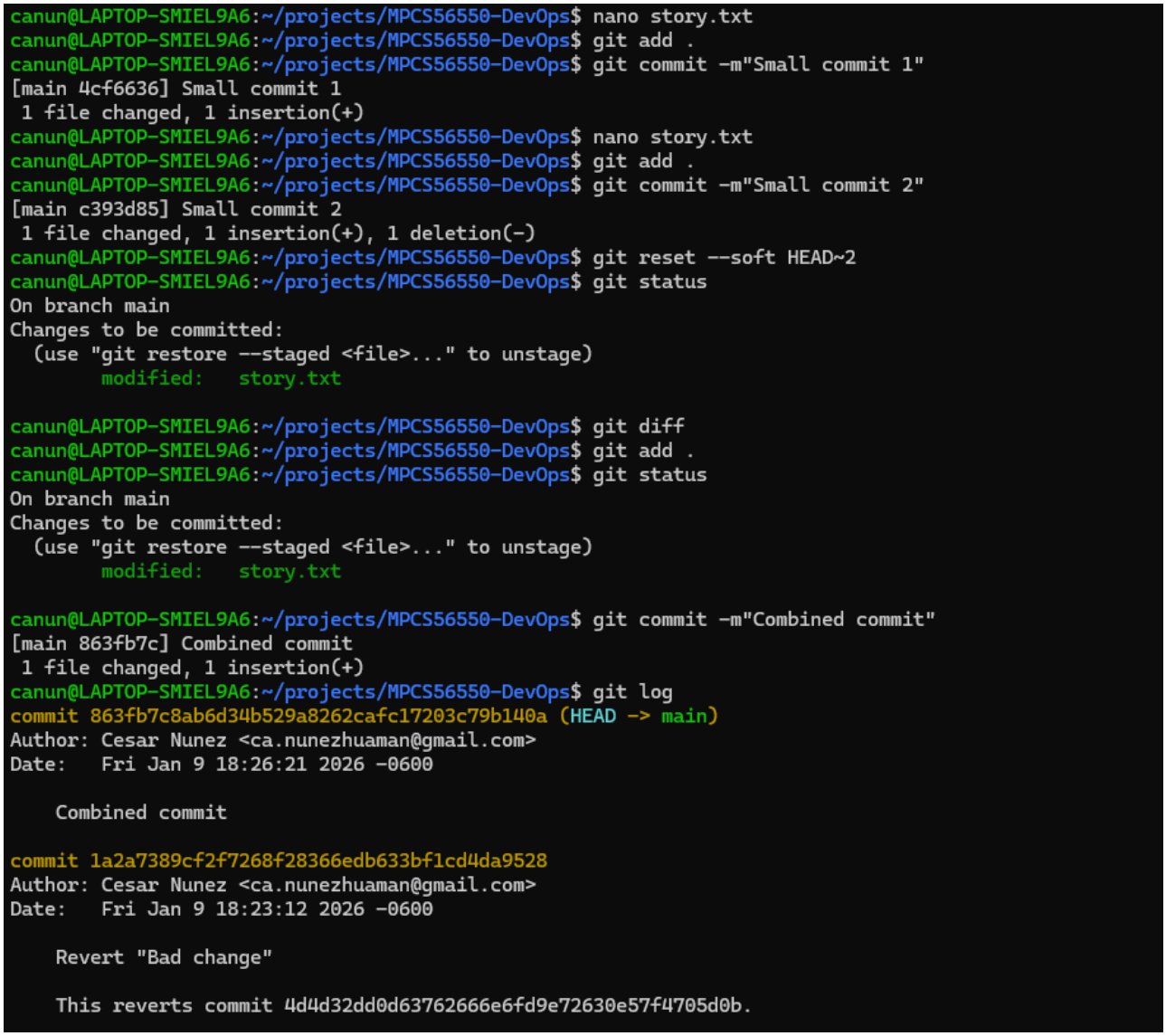
Exercise 3: Git Revision and History

Deliverables

- Screenshots showing the `git log` and one example of each undo operation:

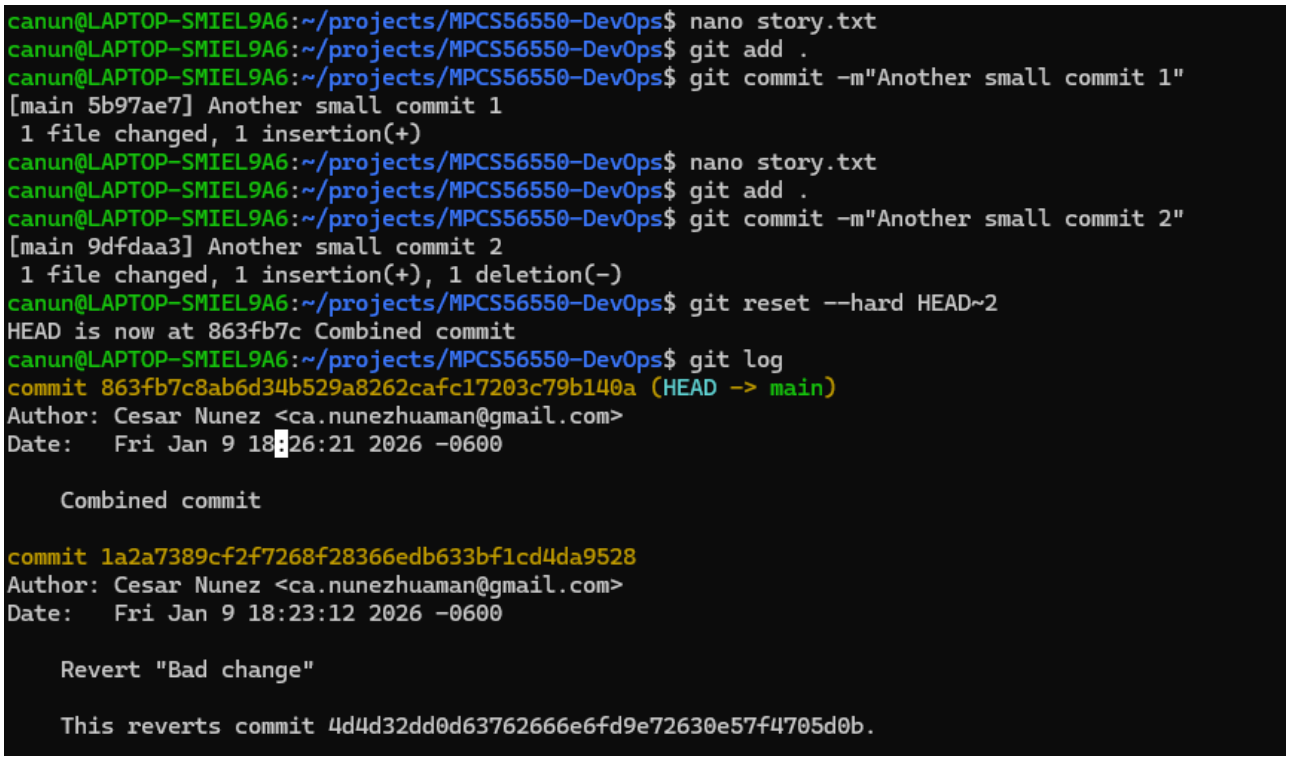
```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ git log
commit 1a2a7389cf2f7268f28366edb633bf1cd4da9528 (HEAD -> main)
Author: Cesar Nunez <ca.nunezhuaman@gmail.com>
Date:   Fri Jan 9 18:23:12 2026 -0600

    Revert "Bad change"

    This reverts commit 4d4d32dd0d63762666e6fd9e72630e57f4705d0b.

commit 4d4d32dd0d63762666e6fd9e72630e57f4705d0b
Author: Cesar Nunez <ca.nunezhuaman@gmail.com>
Date:   Fri Jan 9 18:22:16 2026 -0600

    Bad change

commit 353f42a297b9ae6936816c8b793c2a26e394b477
Author: Cesar Nunez <ca.nunezhuaman@gmail.com>
Date:   Fri Jan 9 18:16:19 2026 -0600

    5th commit

commit 232cafe2edbac73b23dcbd4e621f218ab590ffa4
Author: Cesar Nunez <ca.nunezhuaman@gmail.com>
Date:   Fri Jan 9 18:16:07 2026 -0600

    4th commit

commit d9dd8274208d871214586d4138075b211768b317
Author: Cesar Nunez <ca.nunezhuaman@gmail.com>
Date:   Fri Jan 9 18:15:44 2026 -0600

    3rd commit

commit 59ac177468ed9ccea290af731a072a65a7714b68
Author: Cesar Nunez <ca.nunezhuaman@gmail.com>
Date:   Fri Jan 9 18:15:24 2026 -0600

    Second commit

commit 5a82824812e2873b5814b8cece4e179566f25dd7
Author: Cesar Nunez <ca.nunezhuaman@gmail.com>
Date:   Fri Jan 9 18:14:06 2026 -0600

    First commit
```

- Before and after screenshots of `git reset --soft` showing changes moved to staging area:



```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ nano story.txt
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ git add .
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ git commit -m"Small commit 1"
[main 4cf6636] Small commit 1
 1 file changed, 1 insertion(+)
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ nano story.txt
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ git add .
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ git commit -m"Small commit 2"
[main c393d85] Small commit 2
 1 file changed, 1 insertion(+), 1 deletion(-)
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ git reset --soft HEAD~2
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   story.txt

canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ git diff
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ git add .
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   story.txt

canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ git commit -m"Combined commit"
[main 863fb7c] Combined commit
 1 file changed, 1 insertion(+)
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ git log
commit 863fb7c8ab6d34b529a8262cafc17203c79b140a (HEAD -> main)
Author: Cesar Nunez <ca.nunezhuaman@gmail.com>
Date:   Fri Jan 9 18:26:21 2026 -0600

    Combined commit

commit 1a2a7389cf2f7268f28366edb633bf1cd4da9528
Author: Cesar Nunez <ca.nunezhuaman@gmail.com>
Date:   Fri Jan 9 18:23:12 2026 -0600

    Revert "Bad change"

    This reverts commit 4d4d32dd0d63762666e6fd9e72630e57f4705d0b.
```

- Before and after screenshots of `git reset --hard` showing commits completely removed:



```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ nano story.txt
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ git add .
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ git commit -m"Another small commit 1"
[main 5b97ae7] Another small commit 1
 1 file changed, 1 insertion(+)
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ nano story.txt
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ git add .
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ git commit -m"Another small commit 2"
[main 9dfdaa3] Another small commit 2
 1 file changed, 1 insertion(+), 1 deletion(-)
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ git reset --hard HEAD~2
HEAD is now at 863fb7c Combined commit
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ git log
commit 863fb7c8ab6d34b529a8262cafc17203c79b140a (HEAD -> main)
Author: Cesar Nunez <ca.nunezhuaman@gmail.com>
Date:   Fri Jan 9 18:26:21 2026 -0600

    Combined commit

commit 1a2a7389cf2f7268f28366edb633bf1cd4da9528
Author: Cesar Nunez <ca.nunezhuaman@gmail.com>
Date:   Fri Jan 9 18:23:12 2026 -0600

    Revert "Bad change"

    This reverts commit 4d4d32dd0d63762666e6fd9e72630e57f4705d0b.
```

- Screenshot of `git reflog` showing the reset operations:

```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps$ git reflog
863fb7c (HEAD -> main) HEAD@{0}: reset: moving to HEAD~2
9dfdaa3 HEAD@{1}: commit: Another small commit 2
5b97ae7 HEAD@{2}: commit: Another small commit 1
863fb7c (HEAD -> main) HEAD@{3}: commit: Combined commit
1a2a738 HEAD@{4}: reset: moving to HEAD~2
c393d85 HEAD@{5}: commit: Small commit 2
4cf6636 HEAD@{6}: commit: Small commit 1
1a2a738 HEAD@{7}: revert: Revert "Bad change"
4d4d32d HEAD@{8}: commit: Bad change
353f42a HEAD@{9}: checkout: moving from d9dd8274208d871214586d4138075b211768b317 to main
d9dd827 HEAD@{10}: checkout: moving from main to d9dd8274208d871214586d4138075b211768b317
353f42a HEAD@{11}: commit: 5th commit
232cafe HEAD@{12}: commit: 4th commit
d9dd827 HEAD@{13}: commit: 3rd commit
59ac177 HEAD@{14}: commit: Second commit
5a82824 HEAD@{15}: commit (initial): First commit
```

Exercise 4: Git Hooks

Note: I used ChatGPT for this exercise since I'm not proficient with bash scripts. An example of the prompt I used:

```
Give me a git hook script that:
 - Checks if any .txt files are empty and warns the user
 = Prevents commits if the commit would include files with "FIXME" in them
```

Deliverables

- Hook `pre-commit` script:

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

# Get staged files (Added/Copied/Modified/Renamed)
staged_files=$(git diff --cached --name-only --diff-filter=ACMR)

# Track whether we should block the commit
block_commit=0

# 1) Warn if any staged .txt files are empty (empty in the index)
while IFS= read -r f; do
  [[ -z "$f" ]] && continue
  [[ "$f" != *.txt ]] && continue

  # If file is deleted or doesn't exist in index, skip
  if ! git cat-file -e ":$f" 2>/dev/null; then
    continue
  fi

  # Size (in bytes) of staged version
```

```
  size=$(git cat-file -s ":$f")
  if [[ "$size" -eq 0 ]]; then
    echo "WARNING: Staged text file is empty: $f"
    echo "         (Commit allowed, but you may want to add content.)"
  fi
done <<< "$staged_files"

# 2) Block commit if any staged file contains "FIXME" (in staged content)
while IFS= read -r f; do
  [[ -z "$f" ]] && continue

  # Skip binary files and files not present in index
  if ! git cat-file -e ":$f" 2>/dev/null; then
    continue
  fi

  # Search staged content for FIXME
  if git show ":$f" | grep -n "FIXME" >/dev/null 2>&1; then
    echo "ERROR: Commit blocked. Found 'FIXME' in staged file: $f"
    echo "       Remove 'FIXME' before committing."
    block_commit=1
  fi
done <<< "$staged_files"

if [[ "$block_commit" -ne 0 ]]; then
  exit 1
fi

exit 0
```

- Hook `commit-msg` script:

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

msg_file="${1:-}"
if [[ -z "$msg_file" || ! -f "$msg_file" ]]; then
  echo "ERROR: commit-msg hook did not receive a valid commit message
file."
  exit 1
fi

# Use first non-empty, non-comment line as the "subject"
subject="$(grep -v '^[[:space:]]*#' "$msg_file" | sed '/^[[:space:]]*$/d' |
head -n 1 || true)"

if [[ -z "$subject" ]]; then
  echo "ERROR: Commit message is empty."
  exit 1
fi

# 1) At least 5 words
```

```
word_count=$(printf "%s" "$subject" | awk '{print NF}')
if [[ "$word_count" -lt 5 ]]; then
  echo "ERROR: Commit message must be at least 5 words."
  echo "       Got $word_count word(s): \"$subject\""
  exit 1
fi

# 2) Reject all-uppercase messages
# Strip everything except letters and spaces, then compare to uppercase
version.
letters_only="$(printf "%s" "$subject" | tr -cd '[:alpha:][:space:]')"
letters_only_no_space="$(printf "%s" "$letters_only" | tr -d '[:space:]')"

if [[ -n "$letters_only_no_space" ]]; then
  upper="$(printf "%s" "$letters_only" | tr '[:lower:]' '[:upper:]')"
  if [[ "$letters_only" == "$upper" ]]; then
    echo "ERROR: Commit message cannot be ALL UPPERCASE."
    echo "       \"$subject\""
    exit 1
  fi
fi

exit 0
```

- Hook `post-commit` script:

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

# Where to write history (repo root)
repo_root="$(git rev-parse --show-toplevel)"
history_file="$repo_root/commit-history.txt"

timestamp="$(date '+%Y-%m-%d %H:%M:%S')"
msg="$(git log -1 --pretty=%B | tr '\n' ' ' | sed 's/[[:space:]]\+/ /g' |
sed 's/[[:space:]]*$//')"
hash="$(git rev-parse --short HEAD)"

echo "[$timestamp] ($hash) $msg" >> "$history_file"
```

- Screenshot:

```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/homework1/hooks$ git commit
WARNING: Staged text file is empty: empty.txt
        (Commit allowed, but you may want to add content.)
ERROR: Commit message is empty.
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/homework1/hooks$ git commit -m"Message"
WARNING: Staged text file is empty: empty.txt
        (Commit allowed, but you may want to add content.)
ERROR: Commit message must be at least 5 words.
       Got 1 word(s): "Message"
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/homework1/hooks$ git commit -m"Message is a good"
WARNING: Staged text file is empty: empty.txt
        (Commit allowed, but you may want to add content.)
ERROR: Commit message must be at least 5 words.
       Got 4 word(s): "Message is a good"
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/homework1/hooks$ git commit -m"Message is not that long"
WARNING: Staged text file is empty: empty.txt
        (Commit allowed, but you may want to add content.)
[main 705e546] Message is not that long
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 empty.txt
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/homework1/hooks$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        commit-history.txt

nothing added to commit but untracked files present (use "git add" to track)
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/homework1/hooks$ cat commit-history.txt
[2026-01-12 22:29:51] (705e546) Message is not that long
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/homework1/hooks$ echo "FIXME" >> empty.txt
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/homework1/hooks$ ls
README.md  commit-history.txt  empty.txt
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/homework1/hooks$ git add .
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/homework1/hooks$ git commit -m"Message is longer that 5 words"
ERROR: Commit blocked. Found 'FIXME' in staged file: empty.txt
       Remove 'FIXME' before committing.
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/homework1/hooks$ nano empty.txt
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/homework1/hooks$ git add .
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/homework1/hooks$ git commit -m"Message is longer that 5 words"
[main 4105bc0] Message is longer that 5 words
 2 files changed, 2 insertions(+)
 create mode 100644 commit-history.txt
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/homework1/hooks$ cat commit-history.txt
[2026-01-12 22:29:51] (705e546) Message is not that long
[2026-01-12 22:32:28] (4105bc0) Message is longer that 5 words
```

Exercise 5: Remote Repositories and Branches

Deliverables

- Screenshot of `git remote -v` output:

```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/shared-notes$ git remote -v
origin  git@github.com:cesarnunezh/shared-notes.git (fetch)
origin  git@github.com:cesarnunezh/shared-notes.git (push)
```

- Screenshot of `git log --oneline --graph --all` showing local and remote branches:
- Screenshot of your remote repository on GitHub/GitLab showing the commit history

---

Exercise 6: Git Stashing

Deliverables

- Screenshot of `git stash list` showing multiple stashes with descriptive messages:

```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex6$ git stash list
stash@{0}: On main: WIP on feature Z
stash@{1}: On main: WIP on feature Y
stash@{2}: On main: work in progress on feature X
```

- Before and after screenshots showing `git status` when stashing and popping changes:

```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex6$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   project.txt

no changes added to commit (use "git add" and/or "git commit -a")
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex6$ git add project.txt
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex6$ git stash
Saved working directory and index state WIP on main: 971727b First version project.txt
```

```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex6$ git stash save "work in progress on feature X"
Saved working directory and index state On main: work in progress on feature X
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex6$ git stash list
stash@{0}: On main: work in progress on feature X
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex6$ nano project.txt
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex6$ git stash save "WIP on feature Y"
Saved working directory and index state On main: WIP on feature Y
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex6$ git status
On branch main
nothing to commit, working tree clean
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex6$ nano project.txt
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex6$ git stash save "WIP on feature Z"
Saved working directory and index state On main: WIP on feature Z
```

- Screenshot demonstrating successful branch switching with stashed changes:

```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex6$ git add project.txt
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex6$ git stash
Saved working directory and index state WIP on main: 971727b First version project.txt
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex6$ git switch new_branch
Switched to branch 'new_branch'
```

- Example of applying a specific stash from the stash list:

```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex6$ git status
On branch main
nothing to commit, working tree clean
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex6$ nano project.txt
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex6$ git stash save "WIP on feature Z"
Saved working directory and index state On main: WIP on feature Z
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex6$ git stash list
stash@{0}: On main: WIP on feature Z
stash@{1}: On main: WIP on feature Y
stash@{2}: On main: work in progress on feature X
```

Exercise 7: Git Tags and Releases

Deliverables

- Screenshot of `git tag -l` showing all created tags:

```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex7$ git tag -l
v0.9
v1.0
v1.1
```

- Screenshot of `git show v1.1` displaying annotated tag information:

```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex7$ git show v1.1
tag v1.1
Tagger: Cesar Nunez <ca.nunezhuaman@gmail.com>
Date:   Mon Jan 12 15:43:26 2026 -0600

First stable release

commit 1509b726526004f0deb1bf23b3104338ab21866b (HEAD, tag: v1.1, origin/main, main)
Author: Cesar Nunez <ca.nunezhuaman@gmail.com>
Date:   Mon Jan 12 15:42:38 2026 -0600

    Third version

diff --git a/example7.txt b/example7.txt
index 6a3adff..5b3c010 100644
--- a/example7.txt
+++ b/example7.txt
@@ -1,2 +1,3 @@
 First commit
 Second commit
+Third commit
```

- Screenshot of repository state when checked out at a specific tag:

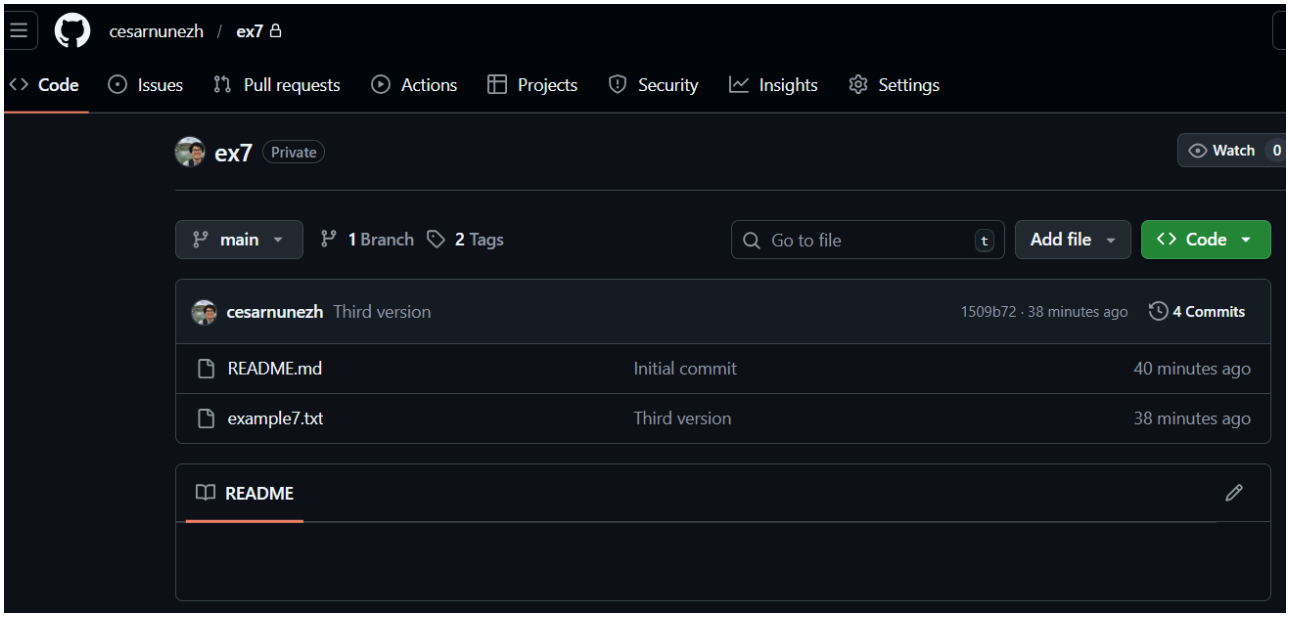
```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex7$ git checkout v1.0
Note: switching to 'v1.0'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 1509b72 Third version
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex7$ ls
README.md  example7.txt
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex7$ cat example7.txt
First commit
Second commit
Third commit
```

- Before and after screenshots of deleting tags (showing `git tag -l output`):

```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex7$ git tag -l
v0.9
v1.0
v1.1
```

```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex7$ git push --delete origin v1.0
To github.com:cesarnunezh/ex7.git
 - [deleted]         v1.0
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex7$ git tag -l
v0.9
v1.1
```

- Screenshot of remote repository showing pushed tags:



- Documentation explaining the difference between lightweight and annotated tags:

```
A lightweight tag does not create a description of the tag, while the
annotated tag have information about the user that created the tag, the
date when the tag was created and the description of the tag. A lightweight
tag only relies on the documentation of the last commit before the creation
of the tag. Annotated tags should be used for releasing the
application/project while lightweight tags should be used for private or
internal presentations.
Source: https://git-scm.com/docs/git-tag
```

Exercise 8: Advanced Merging Strategies

Deliverables

- Screenshot of `git log --graph --oneline --all` showing different merge strategies:

```
canun@LAPTOP-SM1EL9A6:~/projects/MPCS56550-DevOps/ex8$ git log --graph --oneline
*   a7ded7d (HEAD -> linear-feature, main) GIT MERGE
|\
| * 8444d4d Add line 2
| * 0d04427 Add line
| * f5147b5 File creation
* | f11a7f9 Add line 4 in Main
* | 4623245 Add line 3 in Main
|/
* ab8a040 New experimental feature 4
* ee18364 New experimental feature 4
* 6318995 New experimental feature 3
* b1a0a43 New experimental feature 2
* f7e6548 Merge with feature-c
*   110adf7 Merge conflicts solved
|\
| * 04862cd (feature-b) Feature b
* | e7ea344 (feature-a) Feature a
|/
* cd18942 Create project.txt' :
* 9f4cb64 Repo Init
```

- Examples of successful cherry-pick operations with `git log` showing the picked commits:

```
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex8$ cat new_feat.txt
New feature
Change
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex8$ git add .
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex8$ git status
On branch main
You are currently cherry-picking commit 1dfc107.
  (all conflicts fixed: run "git cherry-pick --continue")
  (use "git cherry-pick --skip" to skip this patch)
  (use "git cherry-pick --abort" to cancel the cherry-pick operation)

Changes to be committed:
        new file:   new_feat.txt

canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex8$ git cherry-pick --continue
[main b1a0a43] New experimental feature 2
 Date: Mon Jan 12 16:57:14 2026 -0600
 1 file changed, 2 insertions(+)
 create mode 100644 new_feat.txt
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex8$ git status
On branch main
nothing to commit, working tree clean
canun@LAPTOP-SMIEL9A6:~/projects/MPCS56550-DevOps/ex8$ git log --oneline --graph --all
* b1a0a43 (HEAD -> main) New experimental feature 2
| * 88fb840 (experimental) New experimental feature 4
| * c538aab New experimental feature 3
| * 1dfc107 New experimental feature 2
| * 3b657b9 New experimental feature
|/
* f7e6548 Merge with feature-c
| * 43da15e (feature-c) Changes in c
|/
*   110adf7 Merge conflicts solved
|\
| * 04862cd (feature-b) Feature b
* | e7ea344 (feature-a) Feature a
|/
* cd18942 Create project.txt' :
* 9f4cb64 Repo Init
```

- Explanation to show when to use merge vs rebase vs cherry-pick:

  When to use `merge`: when you want to bring changes from one branch into another while preserving the fact that the changes happended on another branch. Git will create a merge commit and the history will show where branches diverged and came back together. This is usually used for integrating finished features branches into main.

  When to use `rebase`: when you want a unique clean and linear history by replaying your branch's commits on top of another base branch. Rebase does not create a merge commit and keeps individual commits but with different hashes. It's more useful for local or private branches.

  When to use `cherry-pick`: when you want bring specific commits from one branch onto another without mergin the whole branch. This is useful if you have only some commits that are ready, while the rest of the branch is still in progress.
