

Git Command Sequence Reference

The following references all assume you are working in the terminal and have navigated to the base directory for this project capstone.

Create and Checkout a New Branch

Over the next two months of capstone, we will start each task telling you to "Create and Checkout a New Branch". We summarize that process below for your reference. This assumes you have cloned your team's git repository and that you have navigated to your 'capstone' directory before you run any of the commands.

- i. Verify you are on the branch, 'dev'.

```
git branch
```

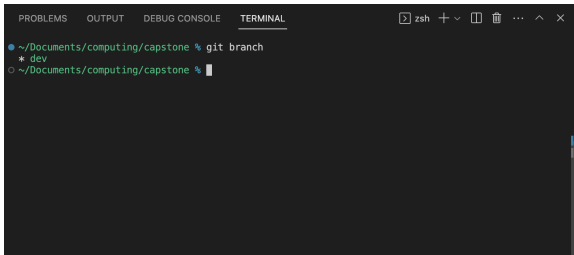


Figure 3.4: good

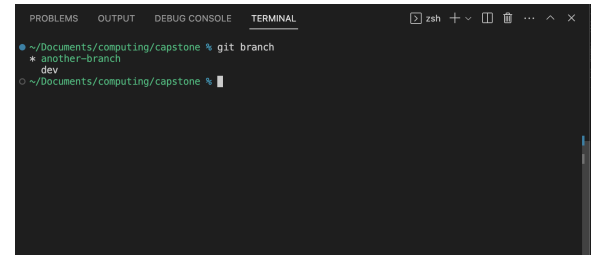


Figure 3.5: bad

If you were not on the branch `dev`, checkout `dev` now.

```
git checkout dev
```

- ii. Verify you do not have any unsaved work.

```
git status
```

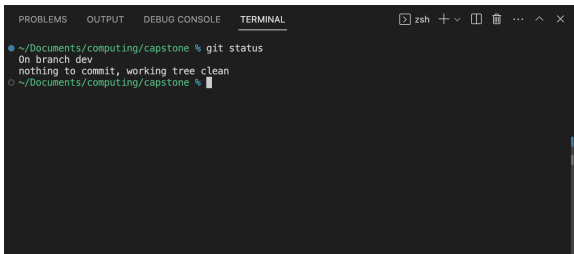


Figure 3.6: No Changes Present

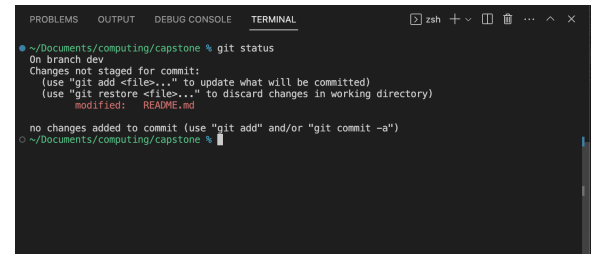


Figure 3.7: Changes to some files present

If you do not have any changes present, then proceed to pull the latest changes. If you do, resolve this by saving your work.

- iii. Pull the latest changes from your remote repository.

```
git pull origin dev
```

- iv. Create and Checkout a New Branch.

```
git checkout -b <new_branch_name>
```

- v. Begin working on your new changes

Stage, Commit, and Push your Changes

Below is the sequence of commands we refer to when we say to "stage, commit, and push your changes".

- i. Verify which files you have changed.

```
git status
```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
~/Documents/computing/capstone % git status
On branch dev
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:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
~/Documents/computing/capstone %

```

Figure 3.8: Only the file you changed included

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
~/Documents/computing/capstone % git status
On branch dev
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    __pycache__/
    app.py

nothing added to commit but untracked files present (use "git add" to track)
~/Documents/computing/capstone %

```

Figure 3.9: Additional files included

If you see additional files or directories present like you can see in Figure 3.9 where the directory `__pycache__` has kept into your changed files you **should not** commit these to your source repository. With only a few exceptions that we will explicitly tell you about, you should only commit files that you explicitly manipulate. To tell git to "ignore" these files you should manipulate your project's `.gitignore` file. To remove the directory `__pycache__` shown in Figure 3.9 I will edit my `.gitignore` by adding the following line and save the file:

```
__pycache__
```

Rerunning `git status` now shows:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
On branch dev
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    __pycache__/
    app.py

nothing added to commit but untracked files present (use "git add" to track)
~/Documents/computing/capstone % git status
On branch dev
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:   .gitignore

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    app.py

no changes added to commit (use "git add" and/or "git commit -a")

```

Figure 3.10: Only the file you changed included

- ii. Once you are confident in which files you have changed, you can manually inspect your changes using `git diff`

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
modified:   .gitignore

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    app.py

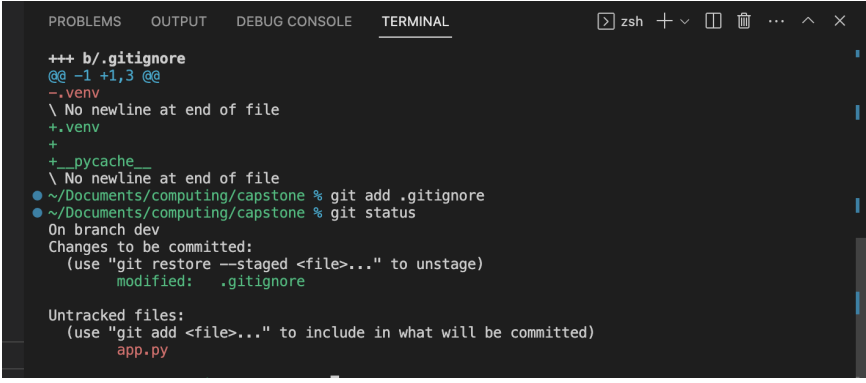
no changes added to commit (use "git add" and/or "git commit -a")
~/Documents/computing/capstone % git diff
diff --git a/.gitignore b/.gitignore
index b694934..95439a9 100644
--- a/.gitignore
+++ b/.gitignore
@@ -1,3 @@
-.venv
\ No newline at end of file
+.venv
+
+__pycache__
\ No newline at end of file

```

Figure 3.11: Inspectable changes to the `.gitignore` file

This useful command line tool assists you in verifying what changes you have been working on are present in your current source branch and will be staged if you stage that file to be committed.

- iii. Add the files you wish to check in to your repository with `git add <filename>`



```

++ b/.gitignore
@@ -1 +1,3 @@
-.venv
\ No newline at end of file
+.venv
+
+__pycache__
\ No newline at end of file
● ~/Documents/computing/capstone % git add .gitignore
● ~/Documents/computing/capstone % git status
On branch dev
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   .gitignore

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    app.py

```


Figure 3.12: The git status after adding a file

You can see here what the outcome is of adding a file, the `.gitignore` file in this case, to your staging area. When you inspect the status afterwards, only those changes included in the "Changes to be committed:" section of the output will be committed to your working branch when you run the following commands. Always be explicit about which files you would like git to stage at any given time. There are shortcuts to add more files at a single time, but you should not need them as you should only be editing a small number of files at any time in this phase of the project. If you find yourself needing to add more files than you can easily and quickly type manually, revisit the previous steps and consider if you are about to commit files you should not be.

- iv. Run `git commit` and add a descriptive commit message. Before you run this command, note the default git editor in the terminal is often `Vim` and many people use this tool (and its acolytes will quickly tell you of its benefits over inferior command line text editors like `Emacs`). If you are familiar with Vim, using that is fine, but if you are not or would prefer to use VSCode. You should follow these steps to setup VSCode as your primary editor.

If you are using Linux or MacOS, open VSCode and the command pallet with `Cmd+Shift+P` or `Ctrl+Shift+P`. Search for "install 'code' command in PATH" and select the option named as such. Once you have installed that command in your PATH, or if you are working on a Windows machine run the following command in the terminal.

```
git config --global code.editor "code -w"
git commit
```

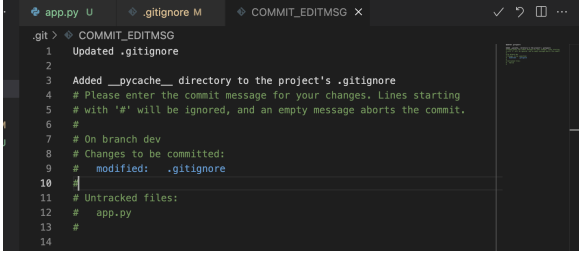


```

Updated .gitignore
Added __pycache__ directory to the project's .gitignore
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch dev
#
# Changes to be committed:
#   modified:   .gitignore
#
# Untracked files:
#   app.py

```

Figure 3.13: Editing a commit message in Vim



```

.git > COMMIT_EDITMSG
1 Updated .gitignore
2
3 Added __pycache__ directory to the project's .gitignore
4 # Please enter the commit message for your changes. Lines starting
5 # with '#' will be ignored, and an empty message aborts the commit.
6 #
7 # On branch dev
8 #
9 # Changes to be committed:
10 #   modified:   .gitignore
11 #
12 # Untracked files:
13 #   app.py
14

```

Figure 3.14: Editing a commit message in VSCode

- v. Once you have created a commit message and saved it, push your changes.

```
git push
```

If this command fails as you have not pushed this branch before, run the command again and add the `--set-upstream` flag to push your branch information and code changes.

```
git push --set-upstream origin <new-branch-name>
```

- vi. Continue working on your branch.

Open a Pull Request and Merge when Ready

When we say to "open a pull request and merge when ready", we are referring to the following steps.

- i. Verify you have pushed all the latest changes from your branch to your remote repository by running `git status` and verifying there is "nothing to commit".

- ii. Open your team's repository in GitHub
- iii. Open the pull requests page
- iv. Select "New Pull Request"
- v. Set the base branch to dev and set the compare branch to the branch you are opening a PR for
- vi. Add a teammate as a reviewer and let them know you have opened a PR
- vii. Once a teammate reviews your code, merge using the GitHub GUI or manually in the terminal
- viii. Resolve any merge conflicts, should they arise (GitHub has a good reference for doing this manually in the terminal, or in a GUI)