# Fundamentals of Git

Illya Starikov

June 30, 2025

These are the show notes of how to use Git for Missouri S&T Satellite Team. This accompany the lecture slides, and each section corresponds to where the words "Demo" appear.

There is some maintenance that we have to take care of before we can begin. It is as follows.

1. Copy the directory `git-tac-toe`[1] to a temporary location (Desktop should work fine).
2. Navigate to said directory in you terminal emulator of choice.
3. Run the commands `git reset --hard 79686d3ae905dc01b1015bd9387dcbadbf6826da` followed by `git branch -D bug-fixes`.

## 1 `Git`ting Good

Let's start with a blank directory[2]. After opening your terminal emulator of choice,

1. Navigate to your desktop directory via `cd`.
2. Make a new directory named `git-tac-toe` for your git repository (`mkdir git-tac-toe`) and change into said directory (`cd git-tac-toe`).

[1] `demo` → `tic-tac-toe`

[2] We don't always have to do this, especially if we already have a git repository somewhere else.

3. Create a git repository within this directory (`git init`).

We are now ready to start using Git! `Initialized empty Git repository in` $\cdots$ or something of that nature should appear. If you receive a warning/error message, changes are git was not set up correctly.

A common convention is to have a `README.md` file in your directory. We will create one now.

1. Create a new file named `README.md` with your text editor of choice (for this demo, I will be using vim).
2. Insert the following text into the file — if you're not familiar with Markdown, the syntax is easy. Check it out here.

```
# Git Tac Toe

This is just a demonstration on the fundamentals of git!
```

3. Save your changes. At this point, we have enough to commit our changes. Add the markdown file with `git add README.md`.
4. Commit your files with `git commit`. A text editor should appear asking you for your commit message (if you've never used Git nor done anything with your environmental variables, it will probably be nano).
   - If you want to change the Git text editor, `git config --global core.editor "EDITOR"`.
   - If you want to change the global text editor, `export VISUAL=vim` followed by `export EDITOR="$VISUAL"`.
5. You commit message should appear like so:

```
Added a read me to describe project
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
#
# Initial commit
#
# Changes to be committed:
#       new file:   README.md
```

Note the commit message isn't particularly long; that's because the intent is very clear.

# 2 Branching And Merging

Now we'll discuss the branching and merging. Please be sure to complete the instructions at the top of this document.

Let's create our game, Git-Tac-Toe.

1. Switch to the branch `game` via `git checkout game`. As you can see, it's already finished! All we have to do is merge!
2. Re-checkout the master branch via `git checkout master`. Merge with `git merge game`.
3. Uh-oh! Merge conflict. Reading the message, we notice it's in `README.md`. We can fix it by deleting one of the two lines; either one is fine.

```
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

4. Re-add `README.md` and commit. Notice the fancy commit message.

```
Merge branch 'game'

# Conflicts:
#       README.md
#
# It looks like you may be committing a merge.
# If this is not correct, please remove the file
#       .git/MERGE_HEAD
# and try again.


# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# All conflicts fixed but you are still merging.
#
# Changes to be committed:
#       new file:   .gitignore
#       modified:   README.md
#       new file:   board.cpp
#       new file:   board.hpp
#       new file:   functions.cpp
#       new file:   functions.hpp
#       new file:   main.cpp
```

5. Compile and run the code (`g++ *.cpp -std=c++14`). Notice after a move, it breaks. Time for debugging.

6. Create a new branch (`git branch bug-fixes`) and checkout said branch (`git checkout bug-fixes`).

7. Open `functions.hpp` and search for "error"; two results should appear.

```
for (const auto& subarray: toFlatten) {
    // This will definitely cause an error
    // result.insert(result.end(), subarray.begin(), subarray.end());
}
```

```
for (int i = 0; i < std::max(vectorT.size(), vectorS.size()); i++) {
    // subtract 1 because size is not 0-indexed

    /* This will also definitely cause an error
    if (vectorS.size() - 1 < i) {
        result.push_back(std::make_tuple(vectorT.at(i), defaultS));
    } else if (vectorT.size() - 1 < i) {
        result.push_back(std::make_tuple(defaultT, vectorS.at(i)));
    } else {*/
        result.push_back(std::make_tuple(vectorT.at(i), vectorS.at(i)));
    //}
}
```

8. Fix these errors one a time, committing in between. Recompile to verify that it works.

9. Checkout the master branch (`git checkout master`) and merge (`git merge bug-fixes`).