Git – Without the complexity

What is Git?

Git is a free and open-source distributed version control system that manages and tracks changes to your code locally on your computer. This cheat sheet features the most important and commonly used Git commands.

Installation

You can download the latest Git version specific to your environment from the links provided below.

- **Download (windows):** https://git-scm.com/download/win
- **Download (macOS):** https://git-scm.com/download/mac
- **Download (Linux/Unix):** https://git-scm.com/download/linux

You can confirm a successful Git installation by running a command in the Command Prompt

```
git --version
```

Setup Global Configuration

In Git, you can set global settings like your full name and email address. These defaults are used as the author information for all your commits.

```
git config --global user.name “Ervis Trupja”
git config --global user.email contact@dotnethow.net
```

Repositories

A repository is a storage location where Git tracks and manages versions of files for a project. It contains the entire history of changes made to the files, allowing for collaboration, version control, and easy navigation through different revisions.

When working with Git it is important to know that you can have local and remote repositories. The first step to working with git is to either use an existing repository from Github (platform and cloud-based service for software development and version control using Git) or you can start your own git repository.

To set up a new Git repository in the current directory, use this command:

```
git init
```

To clone an existing Git repository, simply run the command:

```
git clone /path/to/repository
```

To clone a Git repository from a cloud service like GitHub or Bitbucket, use the command:

```
git clone [remoteUrl]
```

If I have a repository in my GitHub account named **“git-without-the-complexity”**, to clone it I would use the command:

```
git clone https://github.com/etrupja/git-without-the-complexity
```

Git Architecture

Your local Git repository consists of three "trees" maintained by git.

- **WORKING DIRECTORY** - Your current workspace where you edit files. Not tracked by Git until changes are staged.
- **STAGING AREA (INDEX) -** A buffer zone where changes are staged before being committed. You add changes here with **git add**.
- **HEAD** - Contains the history of all committed changes. When you git commit, the changes from the Staging Area are stored here.

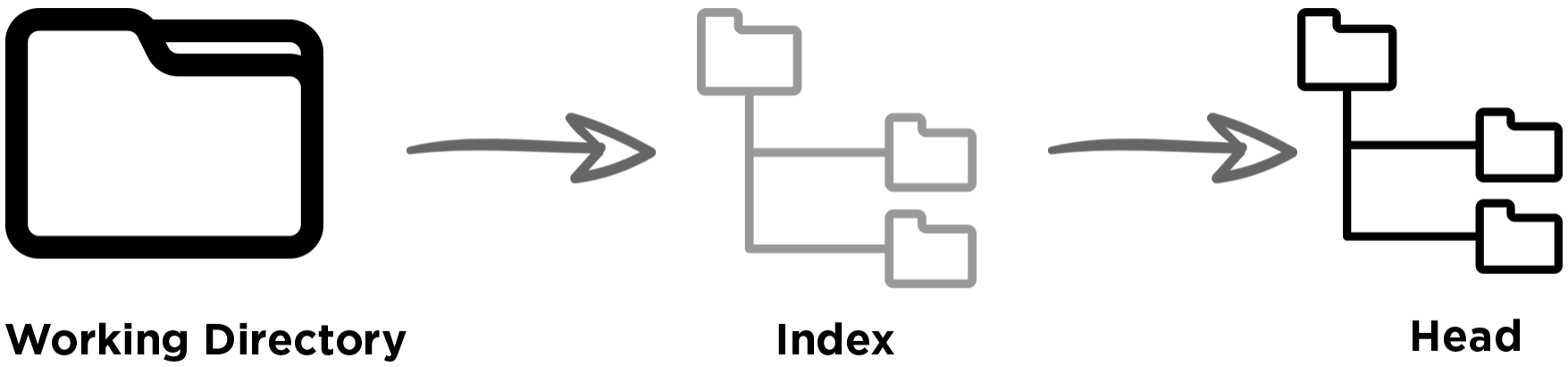

Figure 1 – Git Architecture

Adding Code to Repository

In Git, after making changes to one or more files in your working directory, you need to stage these changes by adding them to the Index before committing them to the repository

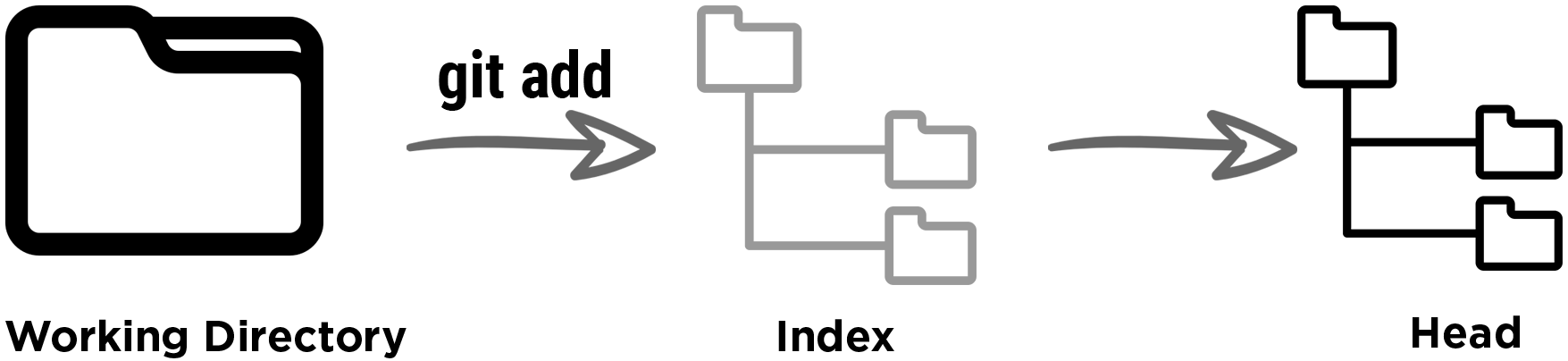

Figure 2 – Add changed to Index with git add

To stage changes for a specific file

```
git add <filename>
```

To stage changes for all files and directories in the current directory, excluding hidden ones and subdirectories.

```
git add *
```

To stage changes for all files and directories in the current directory, including hidden ones and subdirectories.

```
git add .
```

Keep in mind that **`git add`** doesn't permanently save changes in the repository. It just stages them for **`git commit`**, which then stores them in the repository head.

To commit changes to the HEAD, use this command:

```
git commit -m "Your commit message"
```

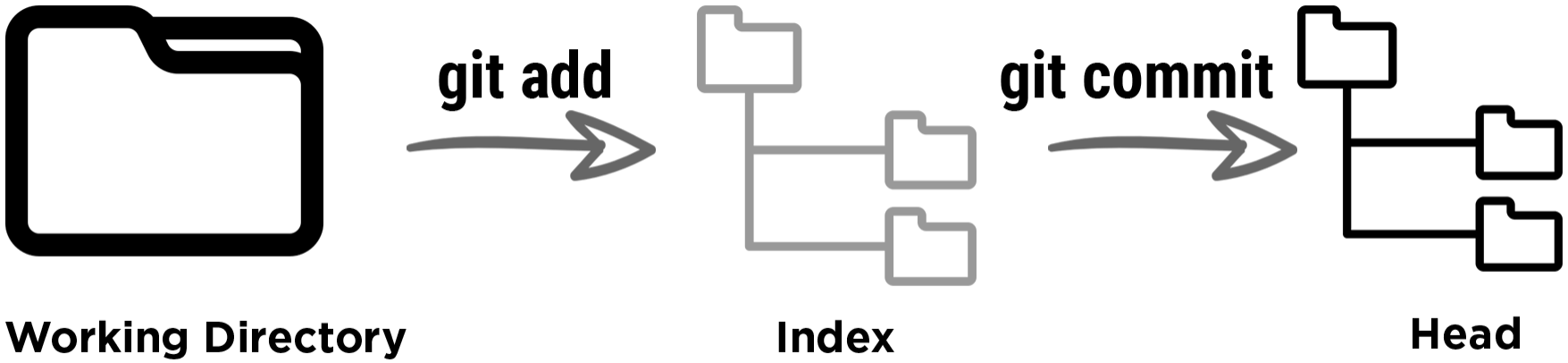

Figure 3 - Adding changes to HEAD with git commit

Now, the changes are committed to the head, but they are not in the remote repository yet (it can be a Github repository etc.)

Pushing Changes to a Repository Branch

Your changes are now in the HEAD of your local working copy. To push these changes to a branch use the command:

```
git push origin [branch_name]
```

So, if you want to push changes to the 'master' branch, use the command:

```
git push origin master
```

If you've created a local repository and want to sync it with a remote repository, you need to link them first. You can do this using the command:

```
git remote add origin <server>
```

Branching

The default branch in a Git repository is the master branch. But it is possible to create other branches (copies) from that branch so you can isolate your work from the rest of the team, or simply isolate the features you are developing from each other.

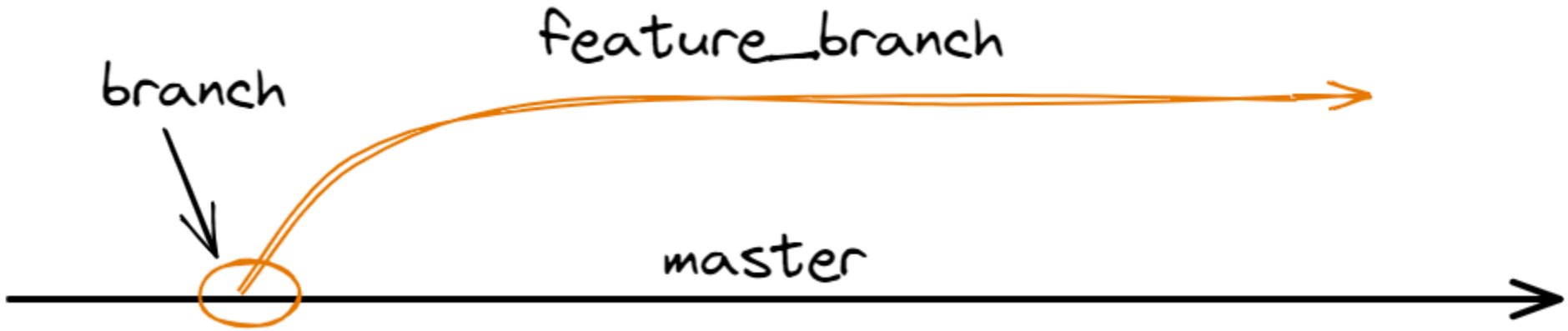

Figure 4 – Creating feature_branch from master branch

After you are done with your changes, you can merge your feature_branch changes back to the master branch. So, moving forward whoever creates another branch from master, will have the latest code, including yours.

To create a new branch, you can use this command:

```
git checkout -b feature_branch
```

To switch back to master branch or to any other branch, you can use:

```
git checkout master
```

You can see that if you use the **-b** flag, a new branch is created, if not then you simply switch to the branch. After you work is done in the feature_branch, you can remove it from git repository using a simple command:

```
git branch -d feature_branch
```

It is very important to know that you can create branches locally, but for others to access them, you need to publish them, so your team members can access them. To publish/push a branch to your remote git repository you can use this command:

```
git push origin <branch>
```

Update & Merge

To download updates from a remote repository into your local repository without merging changes into your current working branch, use the command:

```
git fetch
```

To fetch the latest changes from a remote repository and immediately merge them into your local branch, use the command:

```
git pull
```

This updates your current working environment to match the latest commits in the remote repository.

If you want to merge another branch into your active branch, you can use this command:

```
git merge <branch>
```

So, lets say you checkout and are in the master branch, and to master branch you want to merge the feature_branch changes. In that case, you can use this command:

```
git merge feature_branch
```

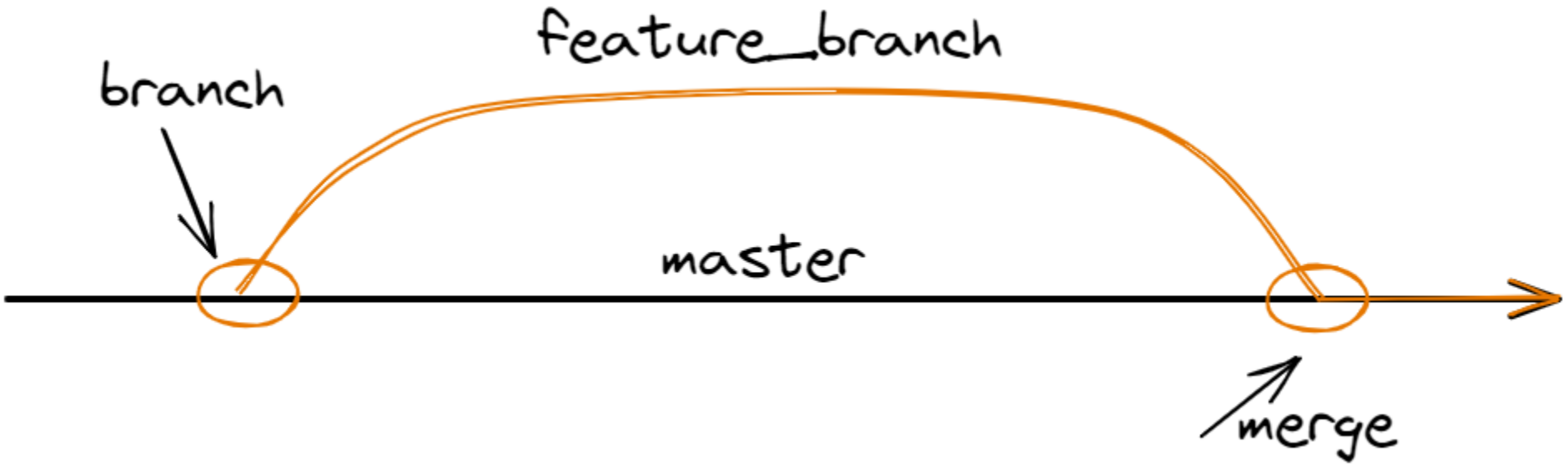

Figure 5 – Merging feature_branch into master branch

When git tries to merge changes, it might not always success, resulting in conflicts. In this case you need to manually resolve the conflicts and decide which code you want to keep from which branch.

This process is called resolving conflicts. You can also check the differences between different branches before you merge the changes. For that you can use this command:

```
git diff <source_branch> <target_branch>
```

Example:

```
git diff master feature_branch
```

Notes

Above we have mentioned some of the commonly used git command, but of course there is more to git. Other than that a lot of GUI tools have been created to make our job easier to work with git.

https://desktop.github.com/ is my favorite tool and I do suggest you go and check it out.

Thank you!

I hope that you foind this file helpful.

I'm Ervis Trupja, a full-stack .NET web developer with a background in mathematics and teaching. Visit my GitHub for my projects and my YouTube for programming tutorials.

I also offer courses on:

- **Udemy:** https://www.udemy.com/user/ervis-trupja/
- **LinkedIn Learning:** https://www.linkedin.com/learning/instructors/ervis-trupja
- **Pluralsight:** https://www.pluralsight.com/authors/ervis-trupja

Excited to share my tech journey with you!