

Git Workshop

Popescu Andrei

 **CC BY 4.0**

Command Line Interface Basics

Note: Directory is equivalent to a folder in the File Explorer / your computer

Given we will be using the CLI often in development, especially with Visual Studio Code and Git Bash, here are some basics.

Change Directory (change which directory your CLI is targeting):

- `cd [directory name]`

List Directory (list the contents of your current directory):

Windows: `dir`

Shell: `ls`

Command Line Interface Basics

Change Drive (change which drive your CLI is targeting):

- [Drive letter]:

List Directory Tree (list the contents of your current directory as a tree):

- tree

Make Directory (create a new directory in the current working directory, note that this does not set your Current Working Directory as the newly create directory):

- mkdir [directory name]

Remove Directory (delete a directory in the current working directory):

- rmdir [directory name]

Command Line Interface Basics

Remove File (delete a file in the current working directory):

- rm [file name]

Clear Console (clear the console so its blank):

Windows: cls

Shell: clear

IP Configuration (check your current IP configuration):

- ipconfig [<options>]

Ping Address (ping a certain address and record 4 ping round trip times):

- ping [address]

Command Line Interface Basics

Exit the console:

- exit
- Ctrl+C

Filepath Basics

Special symbols:

- '.' (period) - Current Working Directory for relative paths
- '..' (double period) - Parent directory for relative paths
- '/' (slash) - subdirectory
- '\' (backslash) - subdirectory in Windows
- 'D:' - Drive letter, usually indicates an absolute path
- '*' (star) - Wildcard, anything goes here
- '"' (quotation marks) - Used whenever a single argument has a space in it

Batch and Shell files

Our operating systems are great and usually come with a way to run multiple commands at once in a single file, either via a batch file (Windows) or shell file (macOS, Linux, most others).

We change the file extension on a text file from .txt to .bat and .sh respectively. This will allow us to execute the files as if we were typing them as commands.

It is important to note Windows comes with two different terminal environments, the Command Prompt and PowerShell. PowerShell can run Command Prompt commands but not vice-versa. Most software (like Visual Studio Code) uses a PowerShell in its terminal window.

Command Line Arguments

Command Line Arguments are a way to pass information from a CLI environment to a program we are executing. They usually involve anything past the first space of a command line. For example, for the java program below, “java -jar filepath/to/jar.jar opt1 opt2 opt3” will have an array of arguments containing [“opt1”, “opt2”, “opt3”].

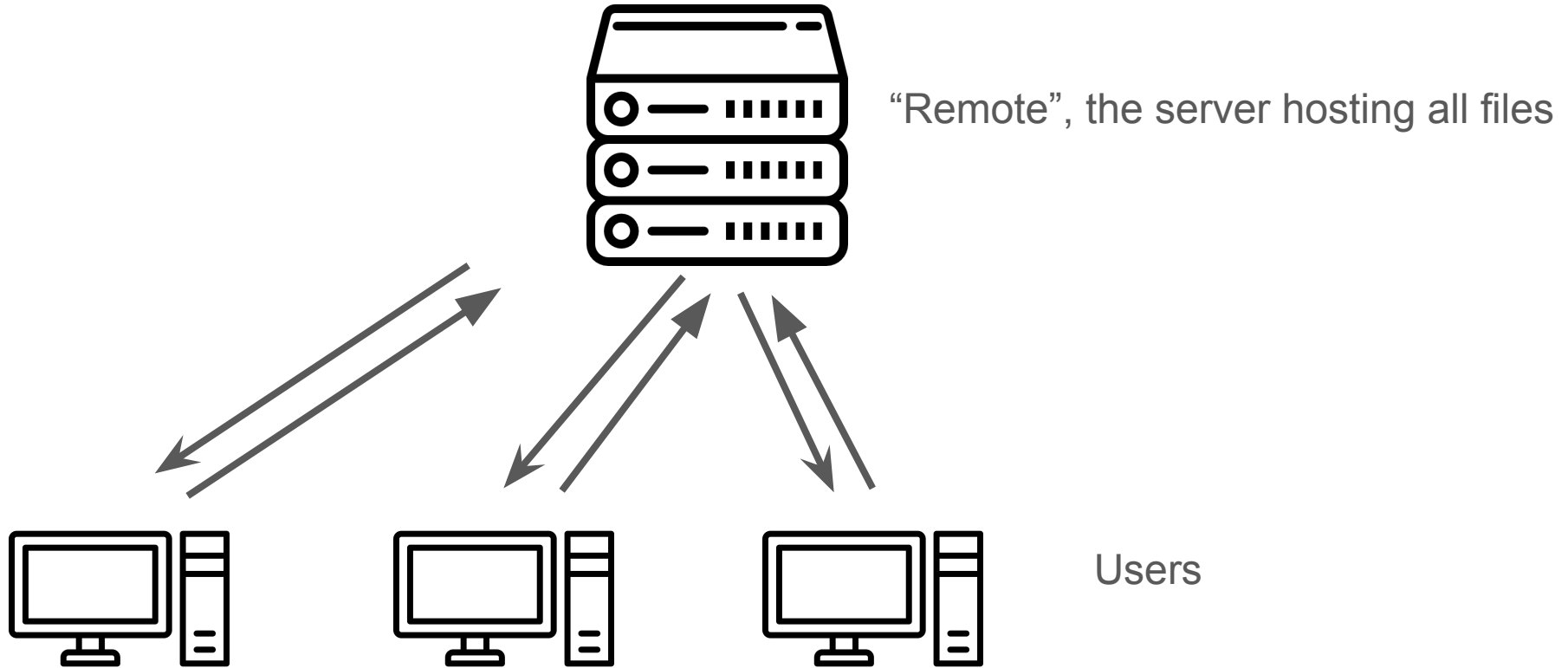
```
1 package org.example;
2
3 public class Main {
4     public static void main(String[] args) {
5
6     }
7 }
```


Git

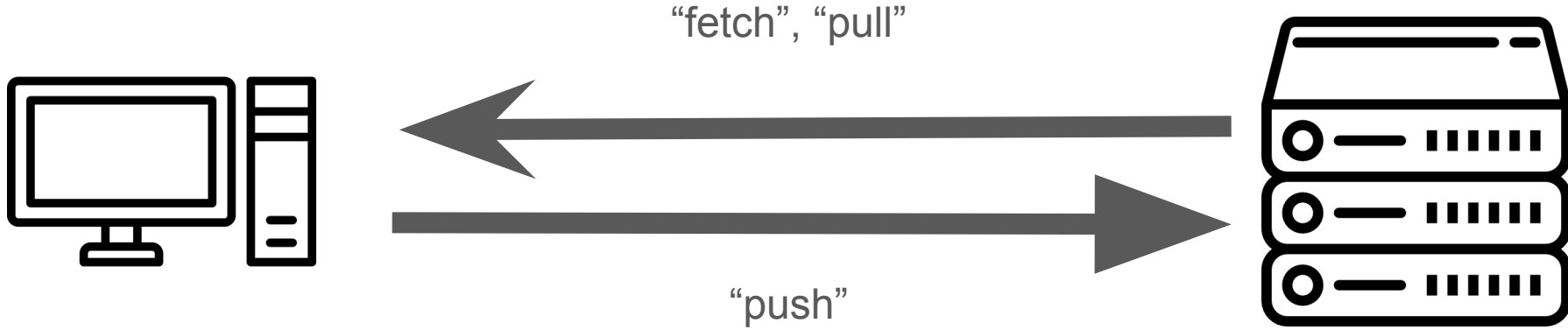
Git is a Version Control System (VCS) or Source Control Management (SCM) system, which allows programmers to synchronize code across different computers easily, without the need to use complex file sharing services or physical media to transfer files. Git is the most widespread system, and services like Github and Gitlab implement Git below the fancy user interface. **We will be using Github.**



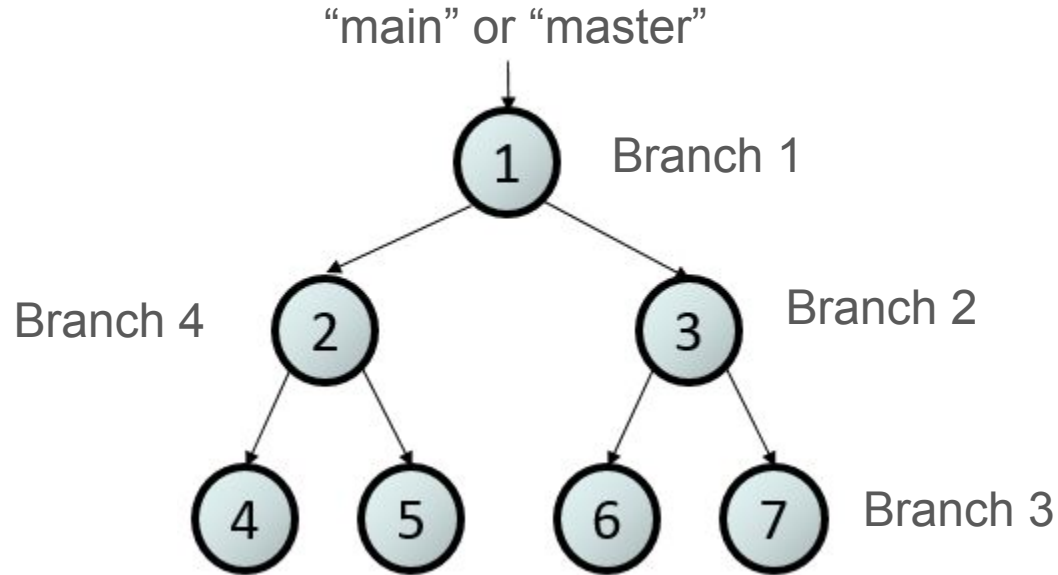
How Git Works - Structure



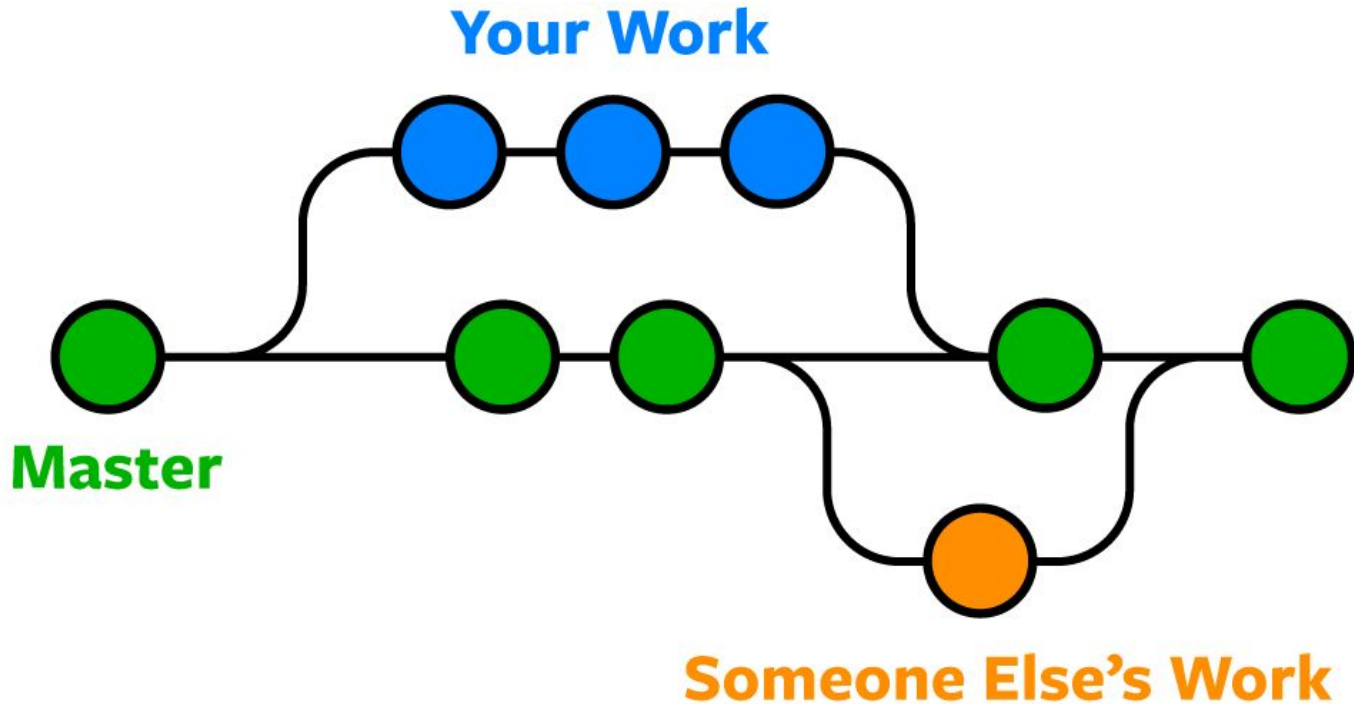
How Git Works - Push, pull and fetch



How Git Works - Branches



How Git Works - Merging and Branching



Git Commands - init

init

Allows the user to initialize a git repository in their Current Working Directory. Everything related to git will be stored in a new directory called “.git”. If you ever wish to completely remove a git system, delete that directory, though you should never have to.

Git Commands - clone

clone [remote]

Allows the user to clone an existing repository from an existing remote. For instance, if we wish to also work on the existing repository situated at <https://github.com/lerdna100/GitWorkshop>, we would then do “git clone <https://github.com/lerdna100/GitWorkshop>”. This will then copy the repository in our Current Working Directory.

Git Commands - Making Changes : add

add [filepath]

Allows the user to add a certain filepath to the git repository's tracker. This file (or the files in a specified directory) will be then tracked for changes, and if they have changed, they will then be able to be synchronized with the remote's.

Note that this only sets a file to be tracked, it does not actually synchronize it. You can modify this file as much as you want without the one on the remote changing as long as you do not commit and push it.

Git Commands - Making Changes : commit

commit -m [commit message]

Allows the user to commit tracked files that have changed since the last commit. A message must always come with a commit to detail what change occurred. In general, the message starts with “add”, “remove”, “modify”, “change”, and so on for the respective operation that happened.

You usually want to commit whenever you have finished implementing a small feature or patch. You should always split changes into commits that make logical sense and are grouped together, even if you are committing them all one after the other.

Git Commands - Making Changes : push

push

Allows the user to finally synchronize all of their commits with the remote's. The user will “push” their changes to the remote, on the branch they are currently working. Now, if anyone fetches and pulls the remote, they will be able to download the new files that have been synchronized.

Git Commands - Downloading Changes : fetch

fetch

Allows the user to “fetch” the remote, pinging it and asking if any changes on the current branch were made since we last checked. Git usually runs this periodically in the background, but it is useful to run before each push to make sure everything is up to date and well on your end.

Git Commands - Downloading Changes : pull

pull

Allows the user to “pull” the remote. This will download any changes that have occurred on your branch since your last pull. It is required to pull before pushing, to ensure the files are always the most recent ones.

Git Commands - status

status

Allows the user to check what files have been modified on their local machine in reference to what the remote has on their current branch. This operation does not affect any files and is simply for user feedback.

Git Commands - diff

diff [commit]

Allows the user to check what files have been modified, by comparing their local files and the files from a certain commit. The commit argument is a unique identifier attributed to each commit, which will be shown how to acquire in the next slide.

Git Commands - log

log

Shows the user all commits on their branch. This is useful for retrieving commit identifiers or just to see the full history (meaning the commit message and otherwise) of all commits on a branch.

```
D:\code\workshops\git>git log
commit 0dc62a2c283c769dcfb4bc61a30d4c33545df7d0 (HEAD -> main, origin/main, origin/HEAD)
Author: Andrei <iernai100@yahoo.com>
Date:   Wed Oct 9 01:44:32 2024 -0400

    initial commit 3

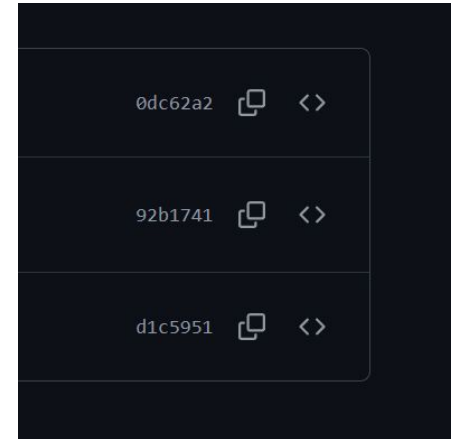
commit 92b17419c11718fdc70703b77d721852b939efbc
Author: Andrei <iernai100@yahoo.com>
Date:   Wed Oct 9 01:42:54 2024 -0400

    initial commit
    Please enter the commit message for your changes. Lines starting

commit d1c5951deab2484dc2d097d2197cebcf9a9f335c
Author: Andrei <iernai100@yahoo.com>
Date:   Wed Oct 9 01:42:07 2024 -0400

    initial commit

D:\code\workshops\git>
```



Git Commands - rm

rm --cached [filepath]

Allows the user to remove a file from the git cache. This is useful if you have updated the .gitignore and no longer want to synchronize a file, as git will not automatically untrack files when the .gitignore updates.

Notice that using this without --cached is equivalent to simply using the “rm” command outside of git, it removes the file(s).

Git Commands - Branches : list

branch --list

Allows the user to see all branches the repository currently has. This command does not modify the repository, it is simply user feedback.

Note that the currently selected directory will be highlighted in green and a green arrow will appear next to it.

Git Commands - Branches : delete

branch --delete

Allows the user to delete a branch if they no longer need it for whichever reason. It is however recommended you keep your old branches, if you ever wish to go back to them. In practice you should rarely use this operation.

Git Commands - Branches : create

branch [name]

Allows the user to create a branch with the provided name. You should do this whenever you want to work on a specific part of the project, in order to minimize merge conflicts when finally merging with the master branch.

Git Commands - Branches : switch

switch [branch name]

Allows the user to switch to a branch with the provided name. Note that this affects your stash, you may want to check if its empty first, and save your current changes to it.

Git Commands - Checkout

checkout [branch name/commit]

Allows the user to switch to a certain branch or a certain commit apart from their current branch. Similar to switch, though you should always use switch unless you need specific functionality from checkout.

Git Commands - merge

merge [branch to merge from]

Allows the user to merge a certain provided branch into the one you are currently on. This will try to implement all of the commits. This operation is one of the more complex operations, and we will see in detail what it can cause if multiple files were edited, and we wish to select a certain specific file or a certain specific part of a file.

Git Commands - Stash

stash

The stash is a sort of temporary storage for when you need to change to a branch shortly but want to save the work you have done without losing it.

Git Commands - Stash : list

stash --list

Allows the user to show the list of existing stashes they have. If the stash is empty, nothing will show up. The stash is a sort of temporary storage for when you need to change to a branch shortly but want to save the work you have done without losing it.

Git Commands - Stash : show

stash --show

Allows the user to see their current stash.

Git Commands - Stash : clear

stash --clear

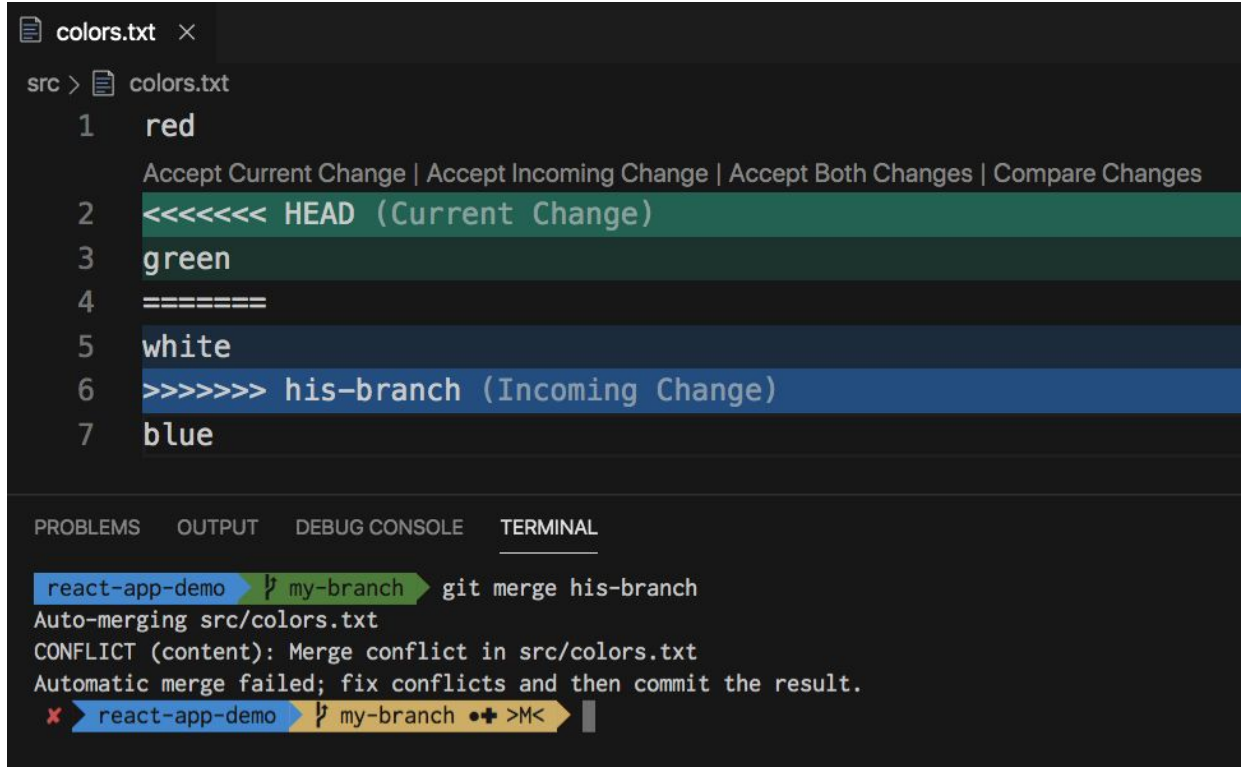
Allows the user to clear their current stash. Note that this operation is irreversible, you cannot get back a stash you have cleared.

.gitignore file

The .gitignore file is a plaintext file situated in the root directory of the git repository (in the same directory as the .git directory). It allows us to exclude certain files from the repository which we do not want to synchronize. Such files include:

- Configuration files
- Sensitive data files (SSH keys, cryptographic salts, local databases)
- Build files (Remember git is for sharing code, not built projects!)
- Other files we don't want others to see, especially if the repository is public

Merge Conflicts - A Quick Overview



The image shows a code editor with a file named `colors.txt` open. The file content is as follows:

```
1 red
   Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
2 <<<<<< HEAD (Current Change)
3 green
4 =====
5 white
6 >>>>>> his-branch (Incoming Change)
7 blue
```

Below the code editor is a terminal window with the following output:

```
react-app-demo > my-branch git merge his-branch
Auto-merging src/colors.txt
CONFLICT (content): Merge conflict in src/colors.txt
Automatic merge failed; fix conflicts and then commit the result.
react-app-demo > my-branch
```

Other Small Points

- We will be using Github for our git system
- Our school offers Github Pro for free
- A quick overview of Github Issues
- Visual Studio Code git integration
- Github Desktop