

# Git + GitHub

An Introduction

# Parts

1. Git - What, Why, Where, How?
2. Git - Basic Workflow
3. GitHub
4. Branches

# 1. Git - What, Why, Where, How?

# What is Git?

- Version control software
- Used everywhere in programming related fields
  - (There is no real alternative these days!)
- **Local** program, independent of GitHub, GitLab, BitBucket etc.
- Integrated in many editors/IDEs

# Why use Git?

- Keep track of changes you make
- Go back to old versions of your code
- Switch between different versions of your code:
  - A "main" version to use everyday
  - A "development" version where you work on a new feature
- Synchronise code between locations
- Collaborate with others

## Where is Git? In repositories!

- Git Repo(sitory) = Root folder of a project
  - Works best with a "Project-centric workflow"
  - You can have as many repos as you want
- By default, everything in the folder is part of the repo
- Can be controlled using a `.gitignore` file
- Git internals are in the (hidden) folder `.git`

# How to use Git?

- **Command Line Interface (CLI)**
  - Uses commands in the terminal, e.g.:

```
git help  
git status  
git commit -am "A Meaningful message"
```

- Harder to learn, but it's good to know the basics
  - The same commands on every computer
- **Graphical User Interfaces (GUIs)**
  - Can do most common tasks
  - Easier to learn
  - GitHub Desktop, VS Code, Pycharm, RStudio...

## 2. Git - Basic Workflow



# Installation and setup

- Download and run installer: <https://git-scm.com/downloads>
- Open the terminal (e.g. on windows `cmd.exe` )
- Set username, email:

```
git config --global user.name "YOUR NAME"  
git config --global user.email YOUR.NAME@example.com
```

# Initializing a repository

- Navigate to your project folder, open a terminal **in this folder**
  - (Example how-to for [windows](#), [mac](#))

- Initialize the repo:

```
git init
```

- Tell git to track all files that are currently in the folder ( `.` ):

```
git add .
```

- Commit the added files

```
git commit -m "Initial Commit"
```

- Git now has a copy of this initial version that you can always come back to!

# Basic workflow

- Make changes (create/edit/delete files)
- Check your changes

```
git status
```

- Stage your changes

```
git add .
```

- Commit your changes

```
git commit -m "This is a meaningful message"
```

- Git has now saved a copy of this second version!

# What is going on?!

- Current state of the repo:

```
git status
```

- List of past commits:

```
git log  
git log --graph --all
```

- Get help:

```
git help  
git help tutorial
```

- Help for an individual command:

```
git commit -h          # Short help  
git commit --help      # Detailed help
```

# Don't commit everything at once!

- Stage only some of your changes

```
git add file1.txt file2.txt
```

- Revert changes you don't want to keep

```
git checkout file3.txt          # Irreversible!!!  
git reset --hard                # Irreversible!!!
```

- Create a file named ".gitignore" to do this automatically

```
# <- Comments start with "#"  
  
file4.txt      #      tell git to ignore "file4.txt"  
*.temp         # *    matches anything in a filename  
**/notes.txt   # **   matches any (sub)directory  
!trackMe.temp  # !    reverses previous gitignore-matches
```

# Going back

- Reverting uncommitted changes

```
git reset --hard          # Irreversible!!!
```

- Reverting committed changes

```
git reset HEAD~1          # Go back 1 commit  
git reset <COMMIT>        # Go back to <COMMIT>
```

- Checking out past commits

```
git checkout <COMMIT>     # View all files from <COMMIT>  
git switch -              # Go back to main branch
```

- Checking out individual files

```
git checkout HEAD~1 file1.txt  # copy file1.txt from previous commit
```

## 2. GitHub

# What is GitHub?

- A website: <https://github.com/>
- A remote location for code tracked by git
  - Share code with others
  - Collaborate on projects
  - Use as personal mirror/cloud
- Similar to e.g. [GitLab](#), [Bitbucket](#), ...




## Create an Account

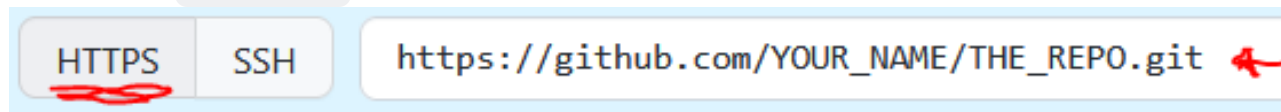
- Go to <https://github.com/signup>
- Fill in your information
- You're ready to go!

# Repositories on GitHub

- Same concept as git repositories on your computer
- Accessible as website or through git
- Private repositories:
  - Only you can see and edit
  - (And collaborators you add)
- Public repositories:
  - Everyone can see
  - Only you can edit
  - (And collaborators you add)

# Creating a repository

- Sign in to [GitHub](#) and click 
- Specify a name
- Select "public" or "private"
- Confirm!
- Select **HTTPS** and copy the link of the repo:



# Pushing a local repo to GitHub

- Open the terminal, navigate to your repository
- Tell git about your GitHub repo

```
git remote add origin https://github.com/YOUR_NAME/THE_REPO.git
```

- Push your changes to GitHub

```
git push -u origin master      # The first time you push something  
  
# (At this point you might be asked to sign in to GitHub!)  
  
git push                      # Subsequent pushes
```

- Check what is going on

```
git status                    # Now includes information about the remote  
git remote -v                 # Shows the connected GitHub repo
```

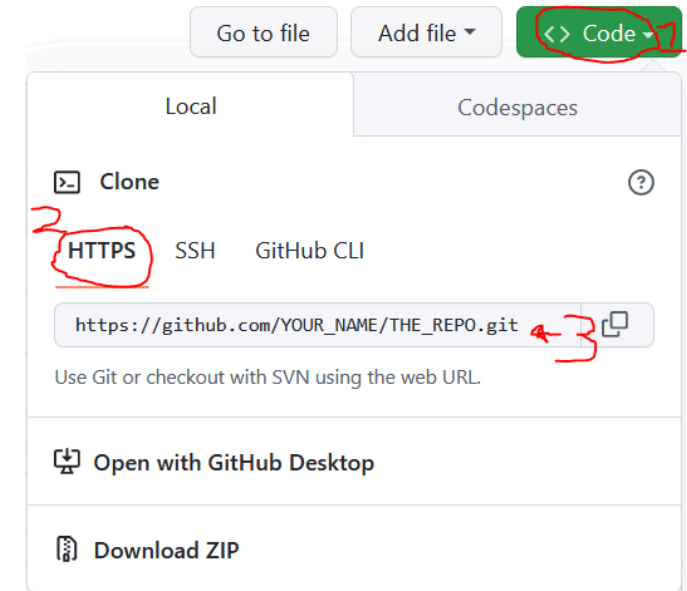
# Cloning a remote repo

- Get the URL of the repo -->
- Open the terminal in the desired parent (!) directory
- Tell git to clone the repo

```
git clone https://github.com/YOUR_NAME/THE_REPO.git
```

- Go into the repo, check things out

```
cd THE_REPO          # (cd = "Change Directory")  
  
git status  
git remote -v
```



# Simple Git/GitHub workflow

- Pull changes from GitHub

```
git pull
```

- Work on the project (**locally**)
- Commit your changes (**locally**)

```
git status  
git add .  
git commit -m "This is a meaningful commit message"
```

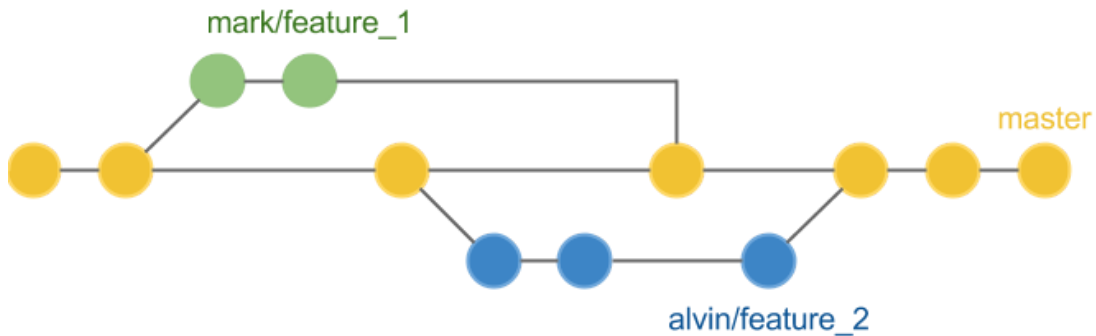
- Push your changes to GitHub

```
git push
```

## 3. Git - Branches

# Why Branches?

- For example, collaborating on a project, you might want:
  - One main "working version" of the code
  - A "personal copy" to work on a new feature
  - Another "personal copy" for someone else



```
git branch <FEATURE BRANCH> # Create a new branch
                               # ...do work...
git merge <FEATURE BRANCH>   # merge the branch
```



# Creating branches

- Create a new branch

```
git branch myFeature    # Creates a branch called "myFeature"  
git branch              # Shows that there is a new branch
```

- Switch to the branch

```
git switch myFeature    # Switches to branch "myFeature"
```

- Commit on the branch

```
git add .  
git commit -m "Implementing my feature"
```

- See what's going on

```
git status              # Includes the name of the current branch  
git branch -v           # Shows a bit more info about all branches  
git log --graph --all   # Shows branches "graphically"
```

# Merging branches

- How to get changes from `myFeature` to `master` / `main` ?
- Switch back to `master` branch

```
git switch master
```

- Merge changes from `myFeature`

```
git merge myFeature
```

- Show result

```
git log --graph --all
```

- (Delete feature branch)

```
git branch -d myFeature
```

## Side-note: Remote tracking branches

- Remote repositories (e.g. GitHub) are tracked by branches

```
git branch -a          # "list both remote-tracking and local branches"
```

- Pulling from a repo ( `git pull` ) is the same as

```
git fetch origin      # Retrieve updates from GitHub  
git merge origin/master # Merge updates into the local branch
```

# Merge conflicts 🤯

- Happen if the main branch and feature branch have **conflicting** changes
- Indicated in the affected files

```
<<<<<< HEAD
1, 2, 3
=====
x, y, z
>>>>>> myFeature
```

- To abort, run

```
git merge --abort
```

- To fix, resolve the conflicts (removing the <<<<< , >>>>> , etc.), then

```
git add .           # Stage all files
git commit --no-edit # Commit with the default merge-commit message
```

# Closing remarks

## Other things to look into

- Pull Requests, Code reviews, Issues, Actions etc. on GitHub
- Squash & merge
- GitHub CLI
- Quickly "stash" away changes: `git stash`
- Find out who wrote what: `git blame`
- Editor integrations, IDEs, GUIs
  - GitHub desktop
  - vscode
  - ...

## General tips

- We just scratched the surface
- Don't hesitate to google or ask AI
- Get familiar with the basic CLI commands
  - Even if you use an IDE/GUI
- Make small commits, commit often
- Use branches for complex changes

**The End**