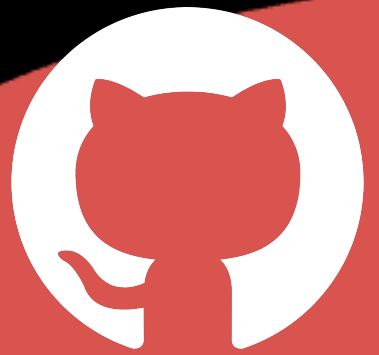
<InfPALS/>

Git I

Fundamentals of Version Control

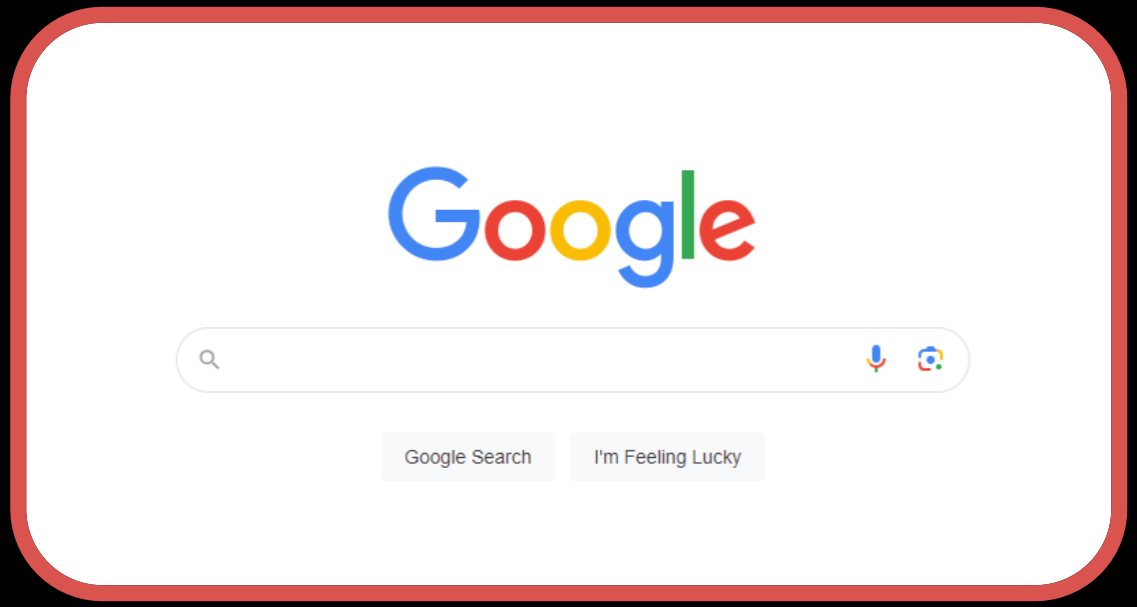


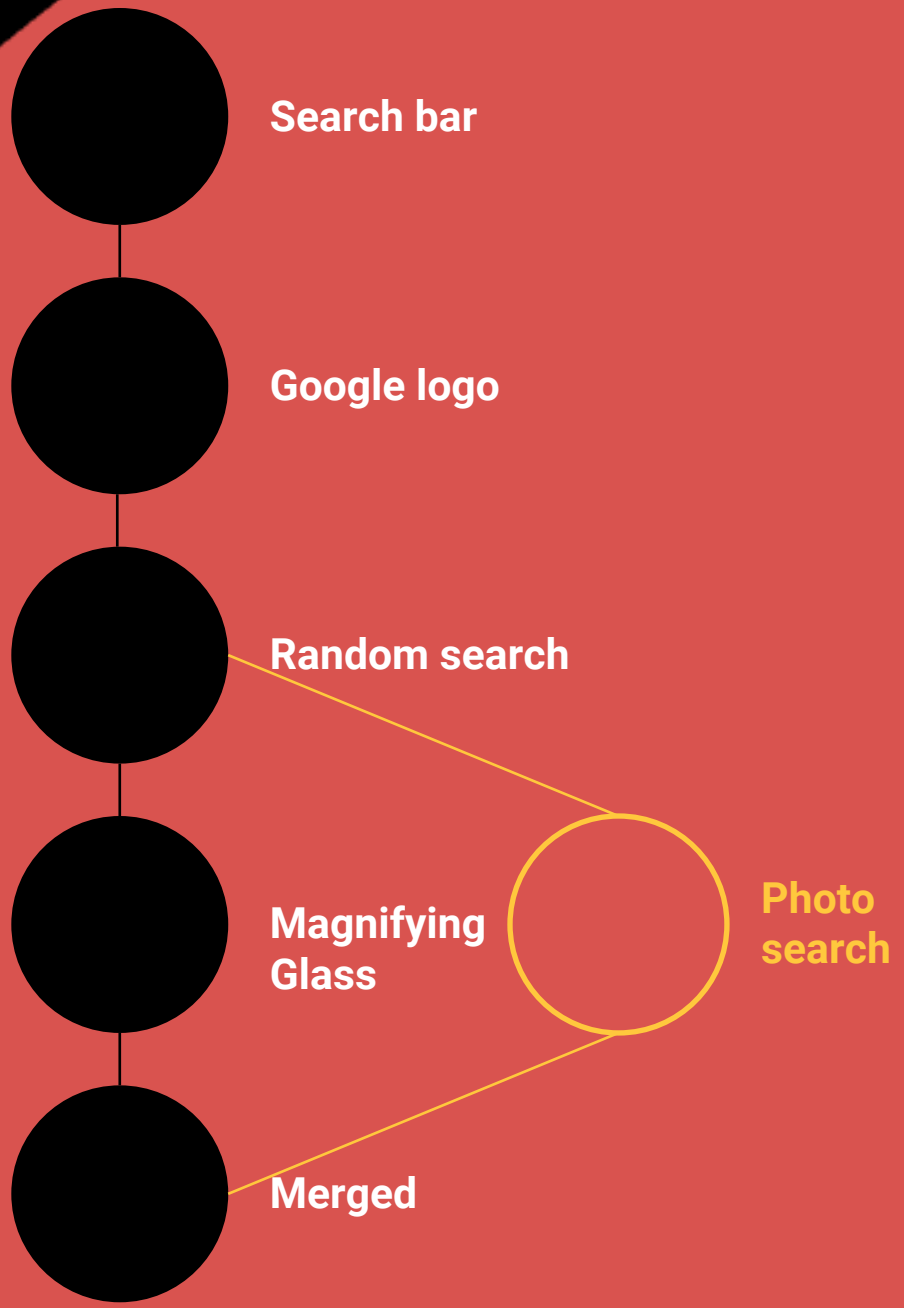
Why do we need it?

- Git: Version control system for tracking code changes & managing project history.
- GitHub: Online platform for hosting Git repositories and collaborating on code.

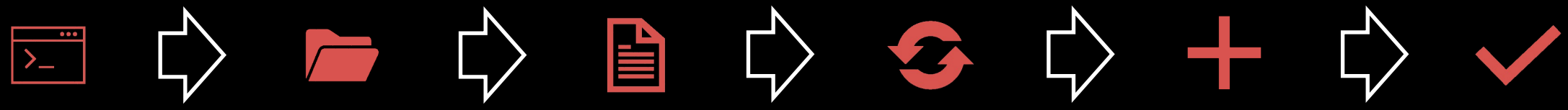
≠

Visualising Git





General workflow



Open terminal

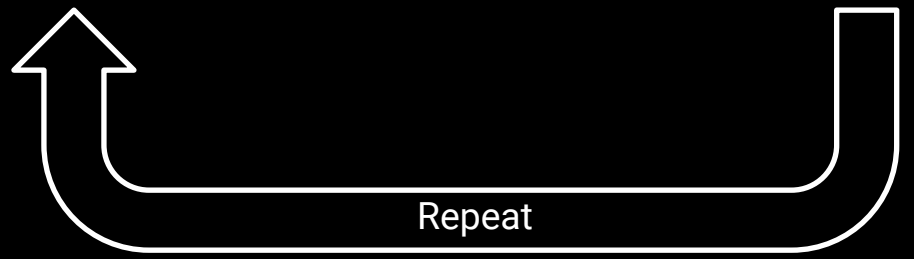
Change directory to the project folder

Initialise git at your folder with git init

Make changes

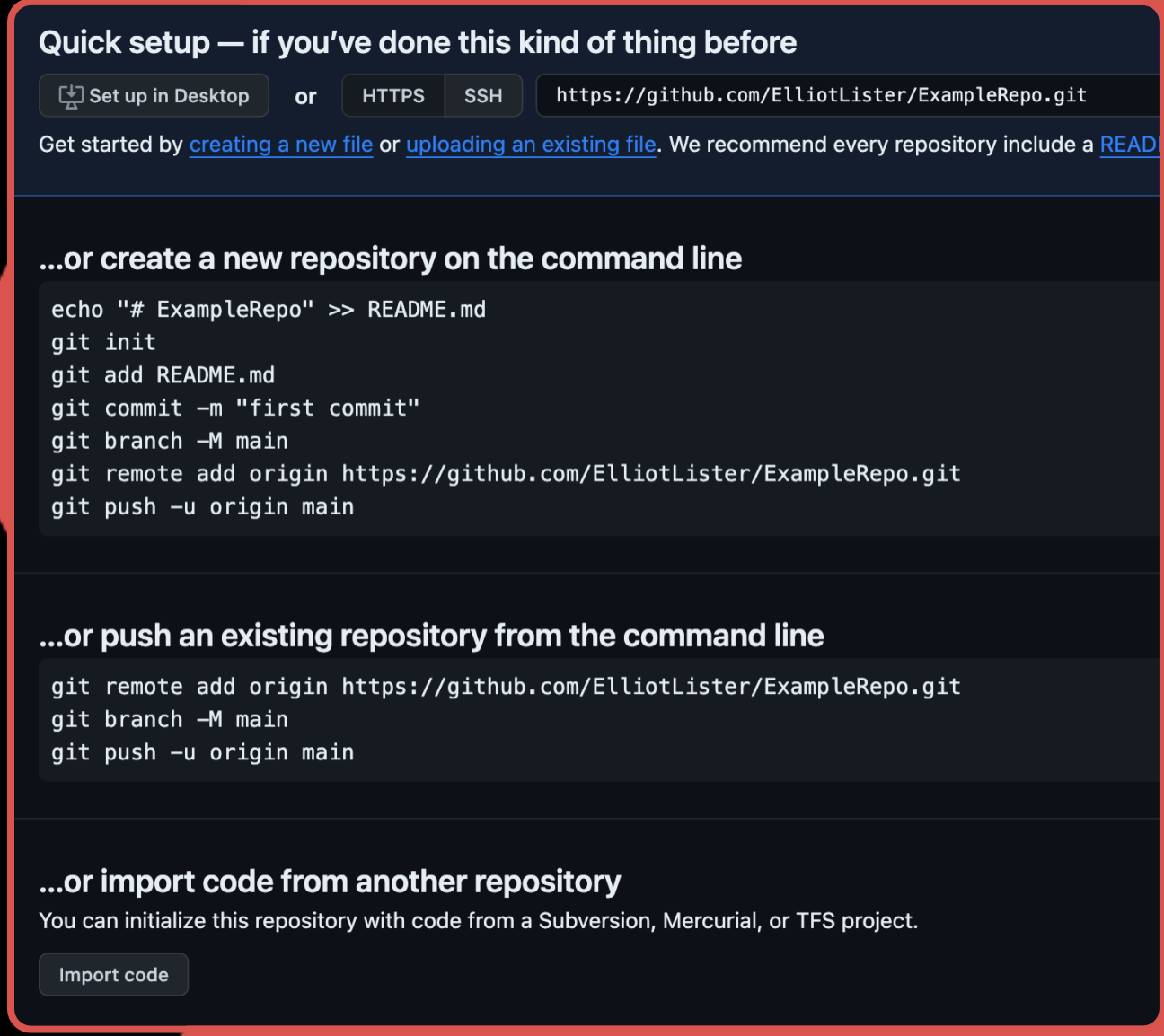
Add files to commit

Commit changes



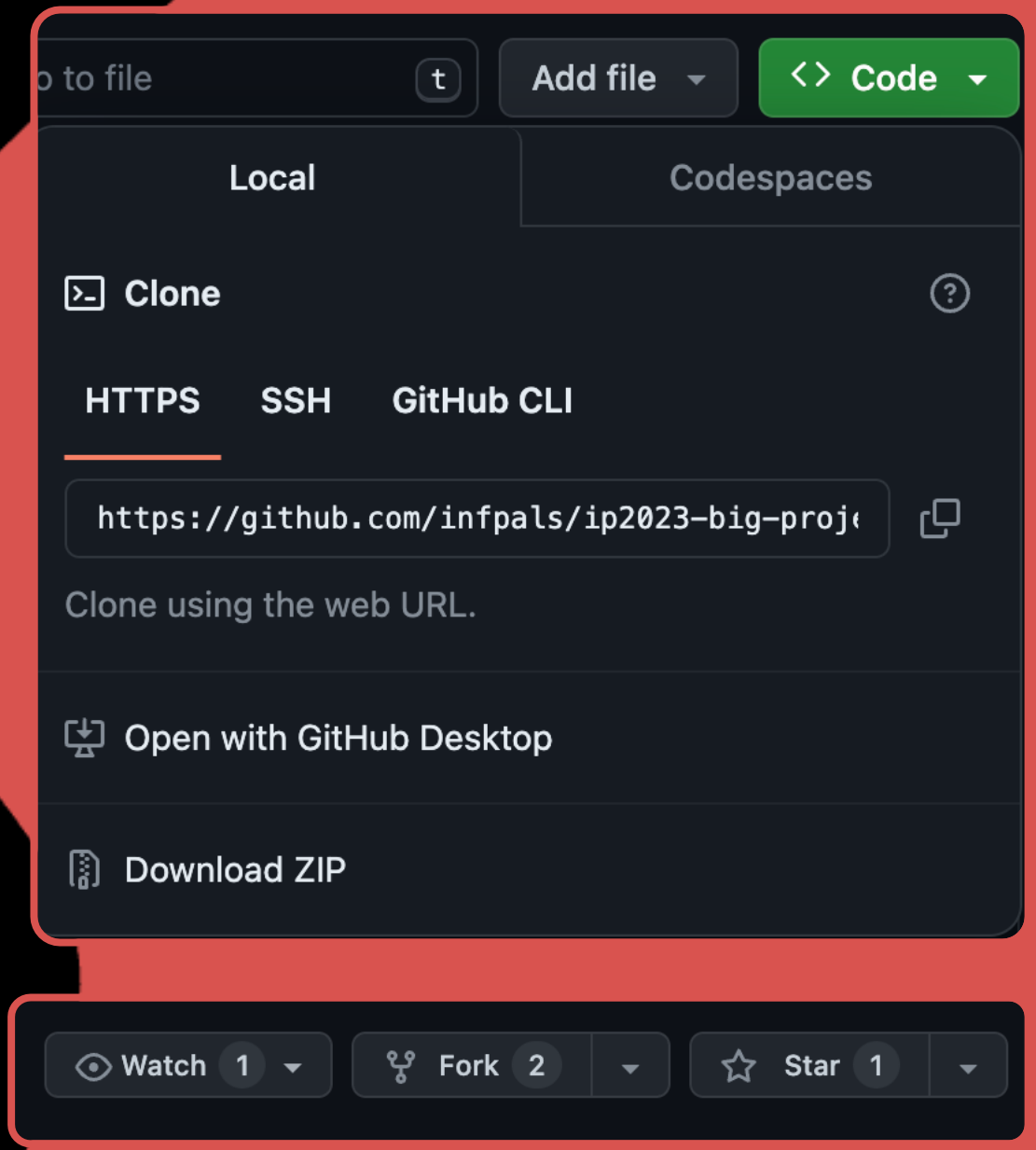
How to link to GitHub

- Sign into GitHub
- Create New Repository
- Depending on whether you have an empty local repo or not, you can create a blank GitHub repo or push your existing local data.
  - Follow the relevant commands for your needs.
- Refresh GitHub
- Start working on your new Repo

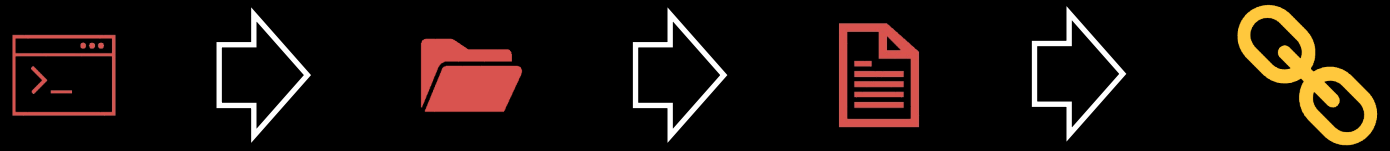


How to link to GitHub

- If the repo already exists on GitHub, you can clone it to your local machine using git clone followed by the HTTPS link.
- If the repo is not one you have access to (e.g. template files for an InfPALS Big Project) you can fork it to your own account and then clone.
  - This way you can push your changes later without any issues with the original.



General workflow with GitHub

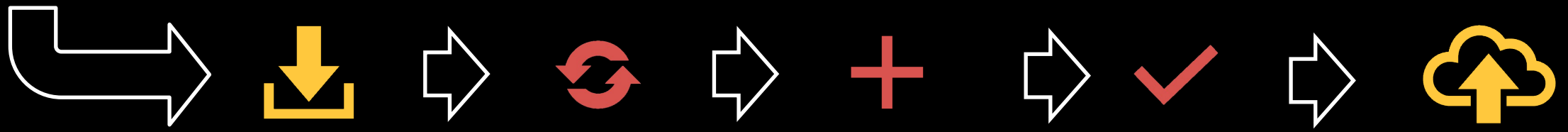


Open terminal

Change directory to the project folder

Initialise git at your folder with git init

Link repo with remote (GitHub)



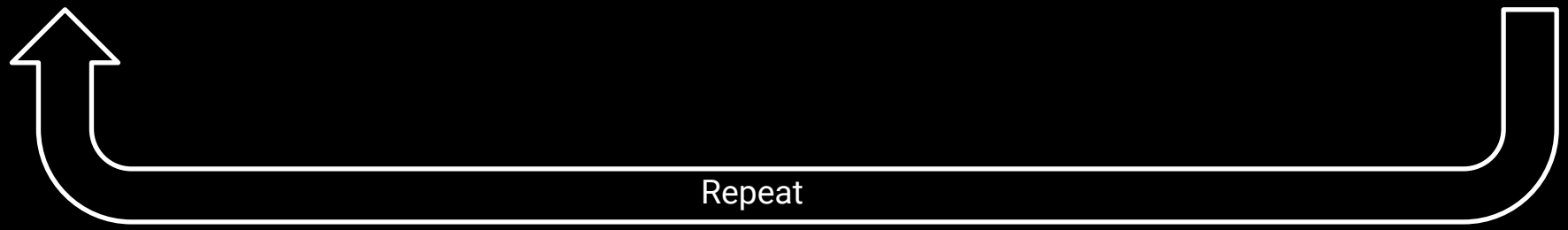
Pull remote changes

Make changes

Add files to commit

Commit changes

Push new changes



Repeat

How can we perform these steps?

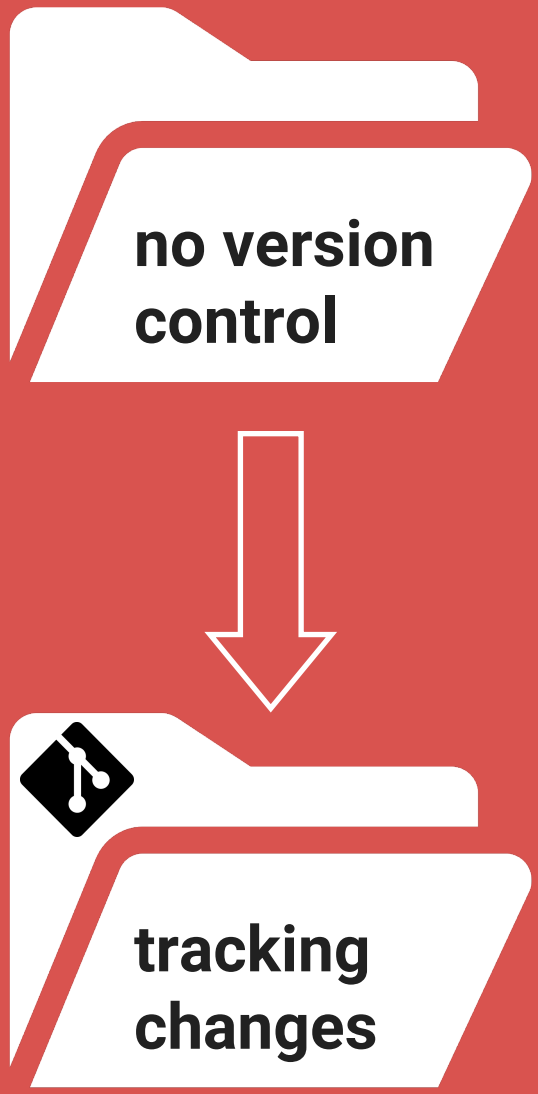
- Now you understand the general workflow, we will explain a bit more detail into each of the commands required.
- Note: Windows does not natively support all terminal commands. Some will work in the command prompt (cmd.exe) and some will work in PowerShell, however it is easier to install git bash which has both a UNIX terminal and git in one package.
  - Download at gitforwindows.org

Let’s open the terminal

If you need any help, do not hesitate to ask!

git init

- Initialises git (tracker of your changes) at a given folder



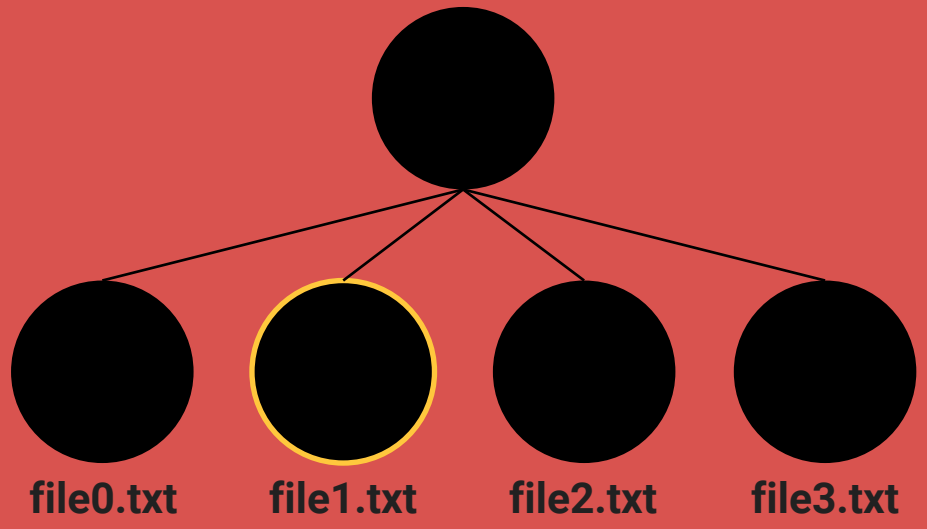
git status

- Modified files: Shows which files you've changed since the last commit.
- Staged changes: Reveals files ready to be included in your next commit.
- Untracked files: Highlights new files not yet added to git or are ignored (e.g. config or data files).
- Uncommitted changes: Alerts you to edits not staged for the next snapshot.
- Branch overview: Briefly mentions the current branch you're working on.
- Potential conflicts: Flags possible issues with unstaged changes or untracked files.

```
On branch main
Your branch is up to date with 'origin/main'.

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:   schedule.html

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

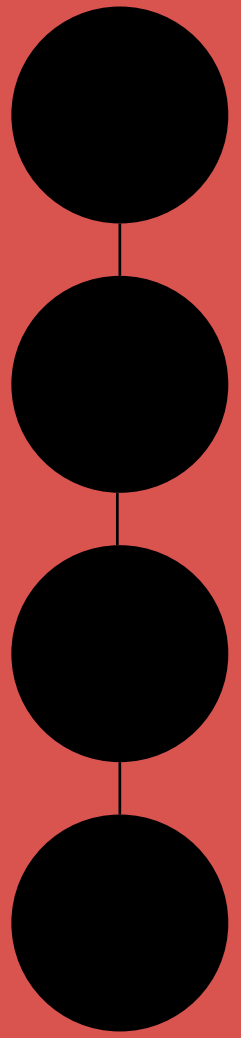
git add

- Selects files for commit: Tells git which changes you want to capture in your next snapshot.
- Stage individual files: **git add file1.txt**
- Stage all modified files: **git add .** captures all changes in your current directory.
- Stage entire folders: **git add <folder>** brings every file inside the folder into the commit.



git commit –m “<message>”

- Captures the current state of your project, creating a checkpoint in your version control history.
- Updates the branch pointer.
- Use the –m flag with an appropriate message, e.g. “Add new stylesheet”
  - If –m isn’t used, nano will open to enter a message. To exit nano type :q and hit return.



Checkpoint 1 created by InfPALS at 22.01.2023 17:05 with message ‘initial commit’

Checkpoint 2 created by InfPALS at 24.01.2023 03:24 with message ‘add slides’

Checkpoint 3 created by InfPALS at 25.01.2023 09:45 with message ‘update slides’

Checkpoint 4 created by InfPALS at 29.01.2023 15:30 with message ‘add solutions’

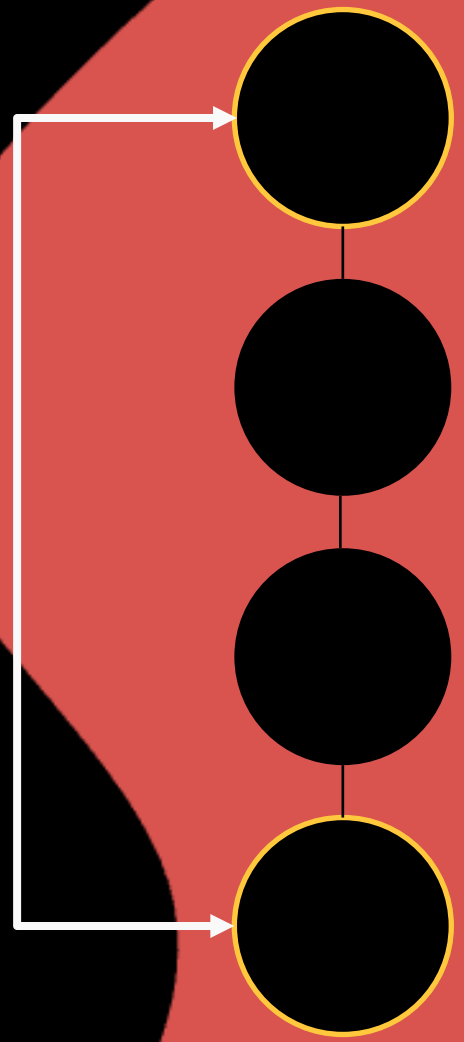
git log

- Shows all commits captured in your repository, starting with the most recent.
- Access commit information like author, date, and commit message.
- Filter log entries by specific authors, dates, keywords, or file changes.
- Each commit is denoted by a hash

```
commit 66f40747f0e3c1559676c7cfc0ccca3addb0f505 (HEAD -
Author: Elliot Lister
Date:   Mon Jan 29 05:18:36 2024 +0000

    Add semester 2 schedule for 2023/24

commit 77f03a8800858dbbbd5d47f939d823c5b885bea5
Author: Elliot Lister
Date:   Wed Sep 27 14:29:26 2023 +0100

    Confirm semester one room bookings

commit 2691821090bc3b8e0c743b1569f8d51da31b5f8c
Author: Elliot Lister
Date:   Wed Sep 27 12:34:29 2023 +0100

    Add detailed schedule

commit 14125e4388e851014aa322cb53de54c0e34419c7
Author: Elliot Lister
Date:   Wed Sep 27 11:34:54 2023 +0100

    Update leader info
```

git diff

- Shows line-by-line alterations between your working directory, staged area, or specific commits.
- Highlights additions, deletions, and modifications made to your code.



Checkpoint 1 created by InfPALS at 22.01.2023 17:05 with message ‘initial commit’

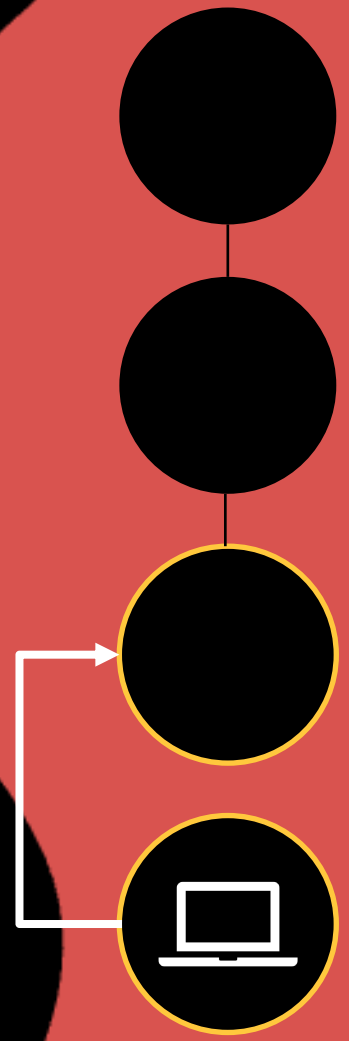
Checkpoint 2 created by InfPALS at 24.01.2023 03:24 with message ‘add slides’

Checkpoint 3 created by InfPALS at 25.01.2023 09:45 with message ‘update slides’

Checkpoint 4 created by InfPALS at 29.01.2023 15:30 with message ‘add solutions’

git reset

- Reverts to a previous commit
- `git reset HEAD` moves the current commit to the previous commit.
- `git reset --soft <commit>` moves the HEAD pointer back to the specified commit, but it keeps all your uncommitted changes in your working directory.
- `git reset --hard <commit>` moves the HEAD pointer back to the specified commit, and it also deletes any uncommitted changes. Think of it as throwing away all your unsaved work.



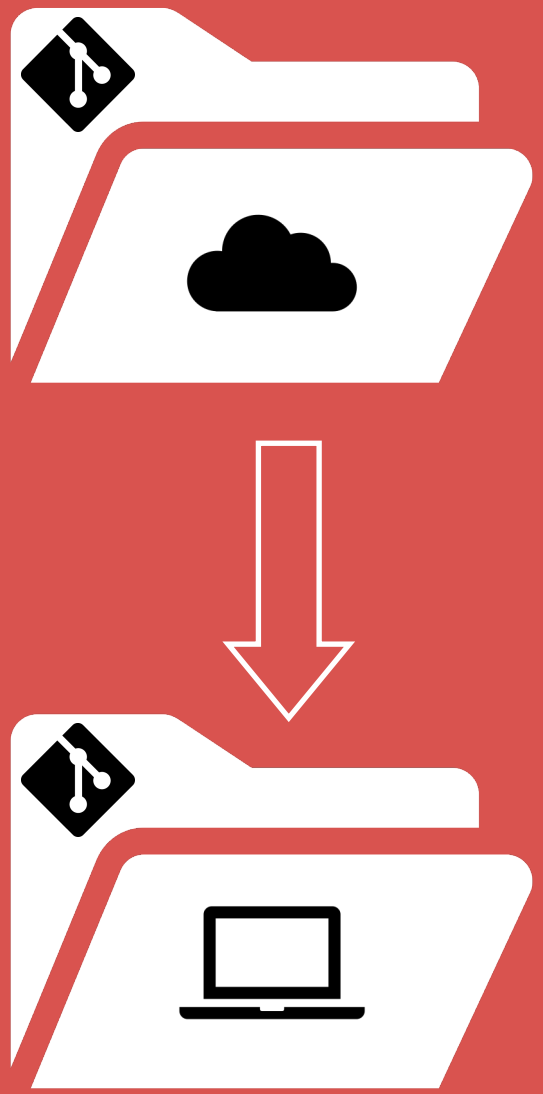
Checkpoint 2 created by InfPALS at 24.01.2023 03:24 with message ‘add slides’

Checkpoint 3 created by InfPALS at 25.01.2023 09:45 with message ‘update slides’

Checkpoint 4 created by InfPALS at 29.01.2023 15:30 with message ‘add solutions’

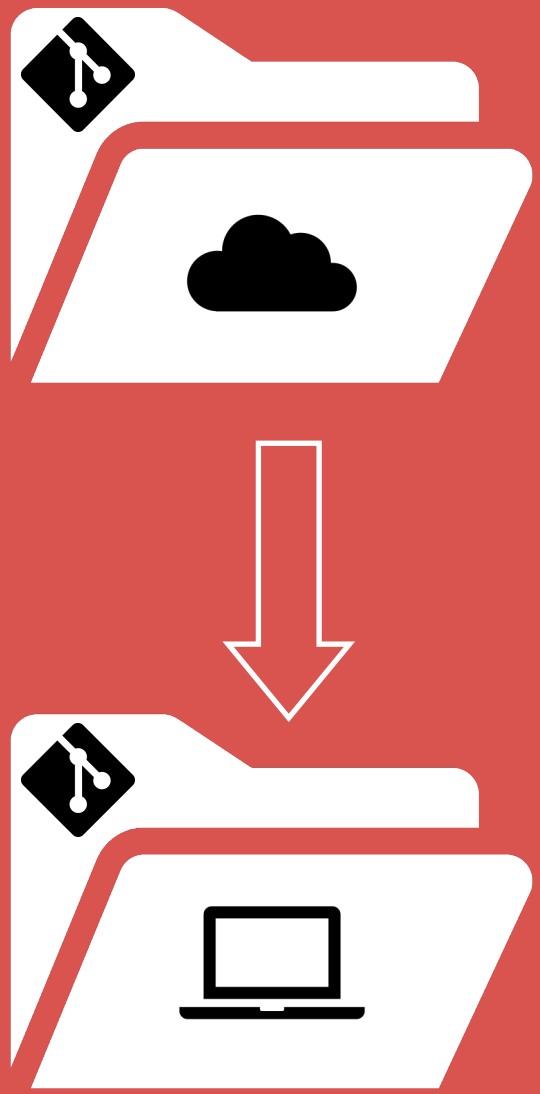
git clone

- Clones a git repo into your local computer with all the historical git data.
- Only run during first initialisation, once link is made, run git pull for future updates.



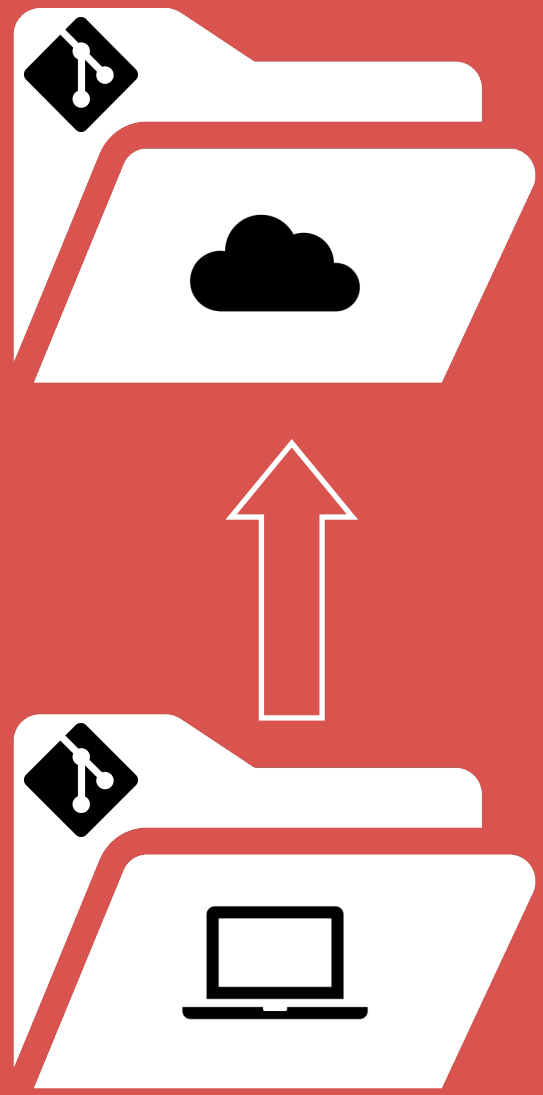
git pull

- Updates your local repo based on the current cloud state of the remote, such as a GitHub repo.
- Requires link to have been made, i.e. through cloning or manually linking.

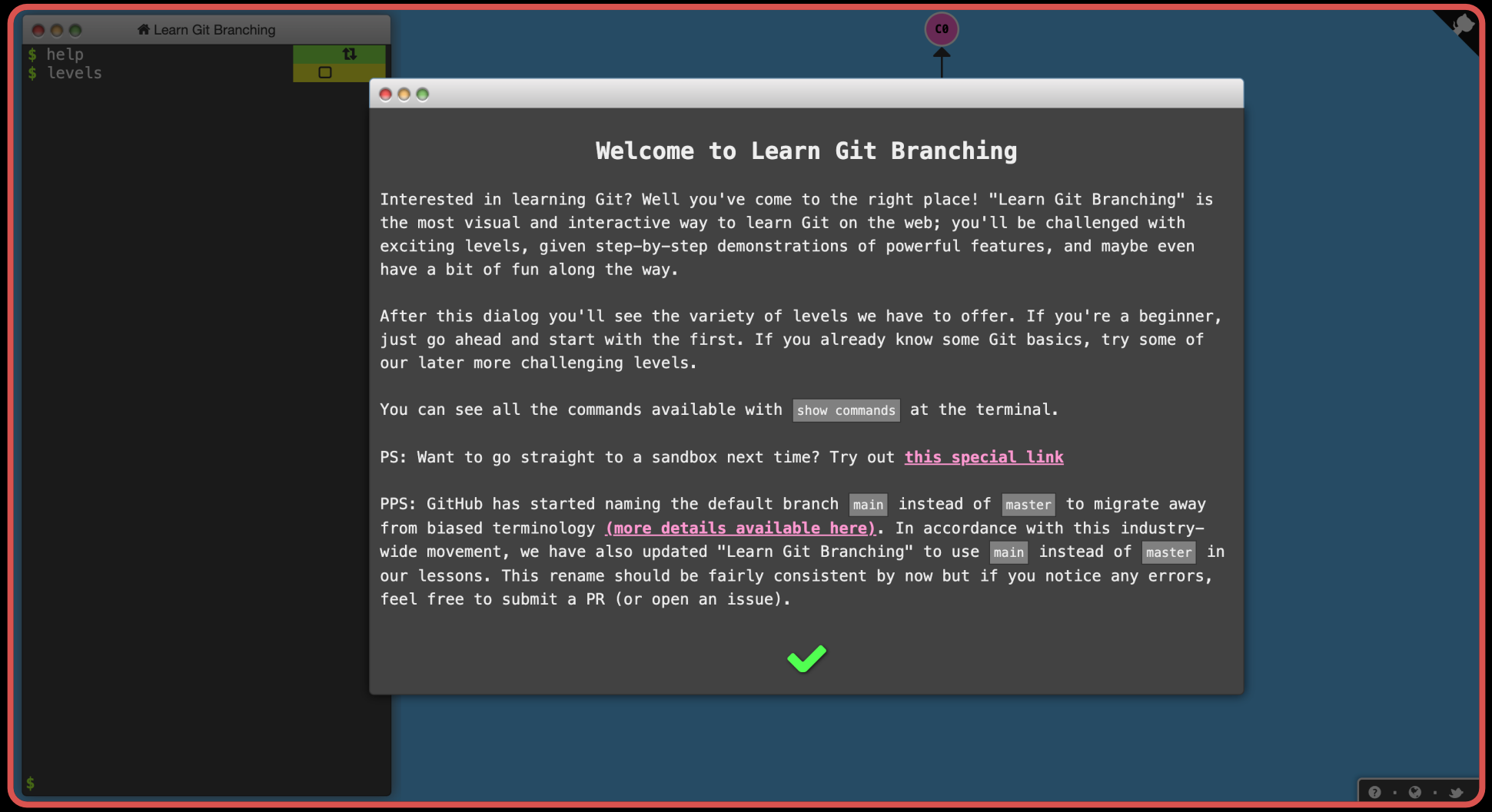


git push

- Pushes file changes to cloud (e.g. GitHub).



Additional resource: learngitbranching.js.org



EXERCISES

Now we are going to put this into practice!

Exercises

1. Open Terminal and navigate yourself to your desktop
2. Create Folder PracticeGit
3. Check the Status of Git in PracticeGit
4. Initialise Git in PracticeGit
5. Create a file Me.txt and create the initial commit with message 'initial commit’
6. Write 'Hello! :) My name is [your name].' into the Me.txt and do another commit with message ‘add my name’
7. Write 'I study <degree>' into the Me.txt, create Colors.txt and do another commit with message 'add my degree' where you only commit changes in Me.txt
8. Do commit for Colors.txt with message 'add Colors.txt file’
9. Print all commits/author/dates/hashes Print difference between initial commit and last commit
10. Create private repository on GitHub Hello-World
11. Write your 3 favourite colours and commit changes with message 'add my 3 favourite colours' and push changes into GitHub repository
12. Clone Materials from the repo for this workshop into your local computer (somewhere where you do not have already initialised git)
13. Print commits done in this repo