Games AI
Lecture 10.2
Git

- IMAGE CONSOLE SANDBOX -

CONSOLE

7 8 9
4 5 6
1 2 3
0 ENTER

STOP STEP PLAY FAST

CONSOLE

ACC 0 | BAK (0) | LAST N/A | MODE IDLE

ACC 0 | BAK (0) | LAST N/A | MODE IDLE

ACC 0 | BAK (0) | LAST N/A | MODE IDLE

STACK MEMORY NODE

ACC 0 | BAK (0) | LAST N/A | MODE IDLE

```
START:
 MOV ACC, DOWN
 ADD 1
 SAV
 SUB 36
 JEZ RESET
 SWP
 JMP START

RESET:
 MOV 0, ACC
 JMP START
```

ACC 0 | BAK (0) | LAST N/A | MODE IDLE

ACC 0 | BAK (0) | LAST N/A | MODE IDLE

```
START:
 SUB 1
 MOV ACC, DOWN
 JEZ RESET
 JMP START
```

ACC 0 | BAK (0) | LAST N/A | MODE IDLE

STACK MEMORY NODE

```
 MOV -2, ACC
 SAV
START:
 MOV UP, ACC
 MOV ACC, RIGHT
 JEZ INCREMENT
RETURN: SWP
 MOV ACC, RIGHT
 SAV
 JMP START
INCREMENT:
 SWP
 ADD 2
 SAV
 JMP RETURN
```

ACC 0 | BAK (0) | LAST N/A | MODE IDLE

```
MOV LEFT, DOWN
MOV LEFT, DOWN
MOV 3, DOWN
MOV -1, DOWN

MOV RIGHT, DOWN
MOV RIGHT, DOWN
MOV 3, DOWN
MOV -1, DOWN
```

ACC 0 | BAK (0) | LAST N/A | MODE IDLE

```
 MOV 21, ACC
 SAV
START:
 MOV UP, ACC
 MOV ACC, LEFT
 JEZ DECREMENT
RETURN: SWP
 MOV ACC, LEFT
 SAV
 JMP START
DECREMENT:
 SWP
 SUB 2
 SAV
 JMP RETURN
```

ACC 0 | BAK (0) | LAST N/A | MODE IDLE

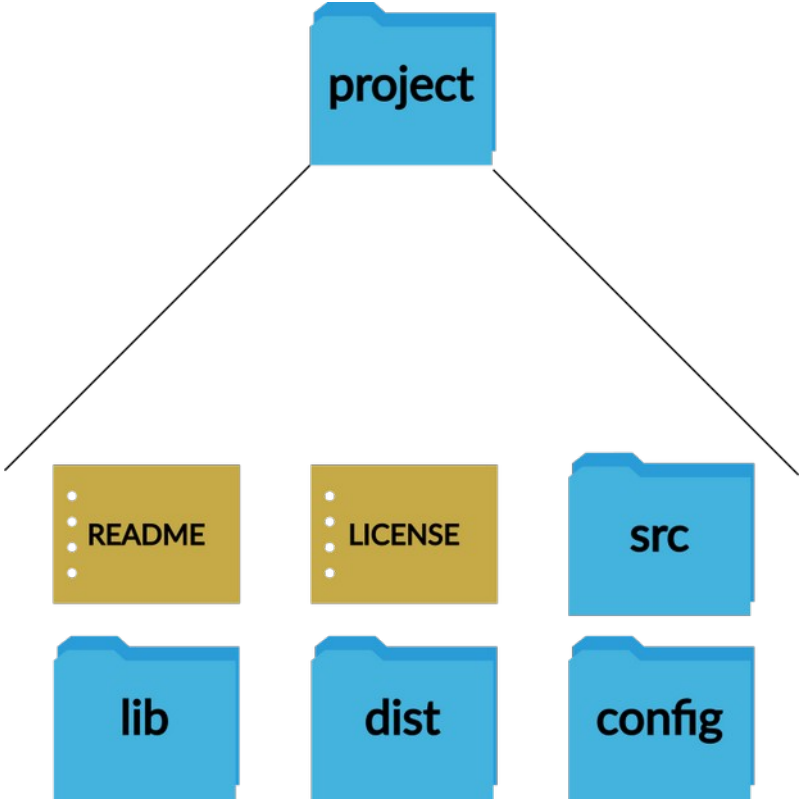
CONSOLE

- Overview
  - Why Git
  - Git Concepts
    - Commits and Branches
    - Working with remotes
  - Basic tasks in git

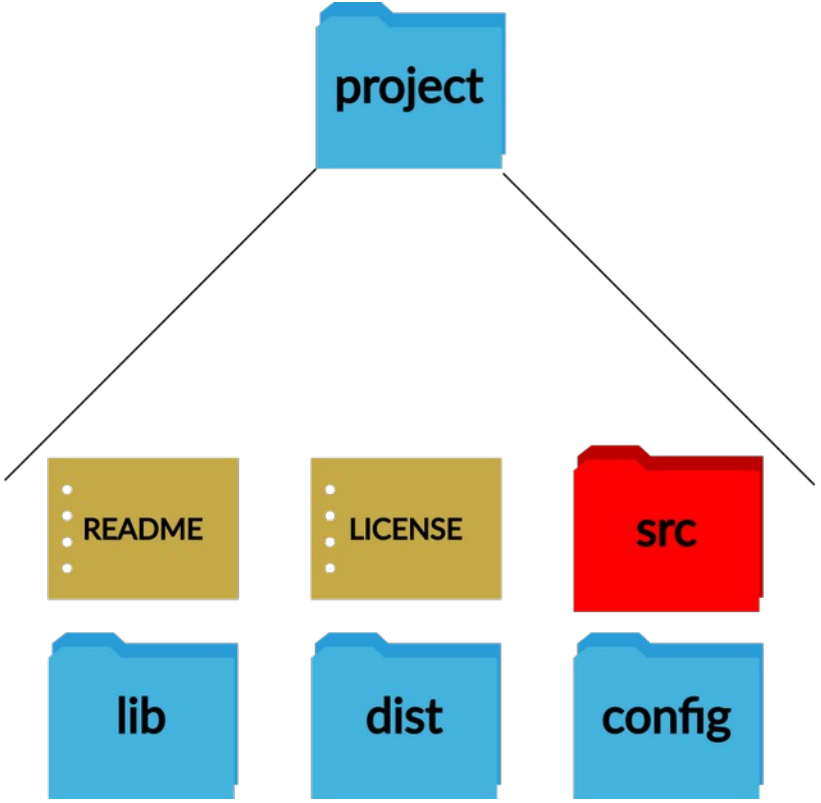
Why Git

- You have a project folder on your computer

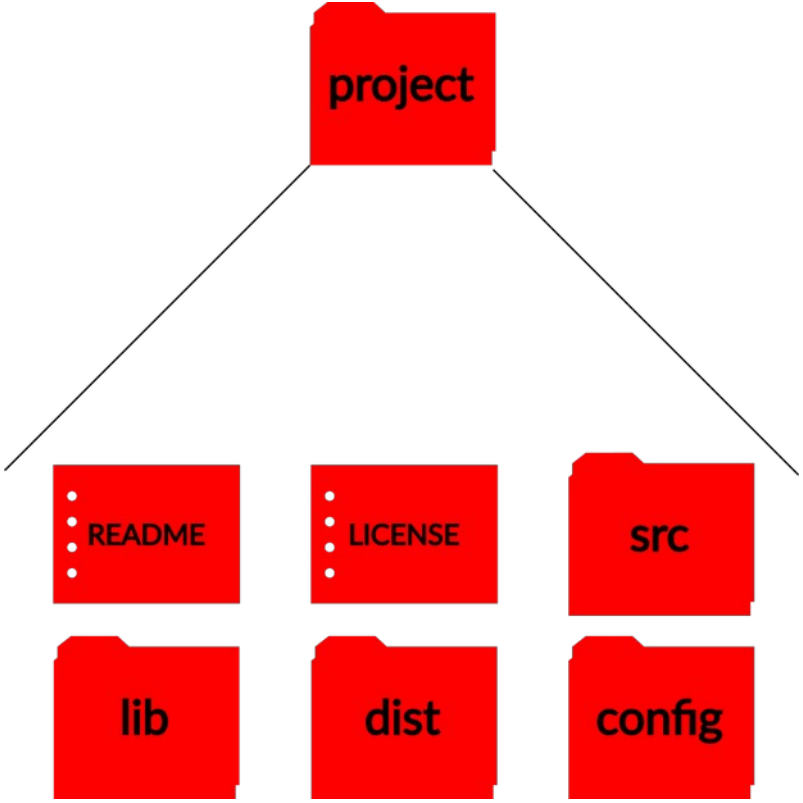
- Inside are the files for your project
  - Source code
  - Libraries
  - Built code
  - Configuration files
  - Assets



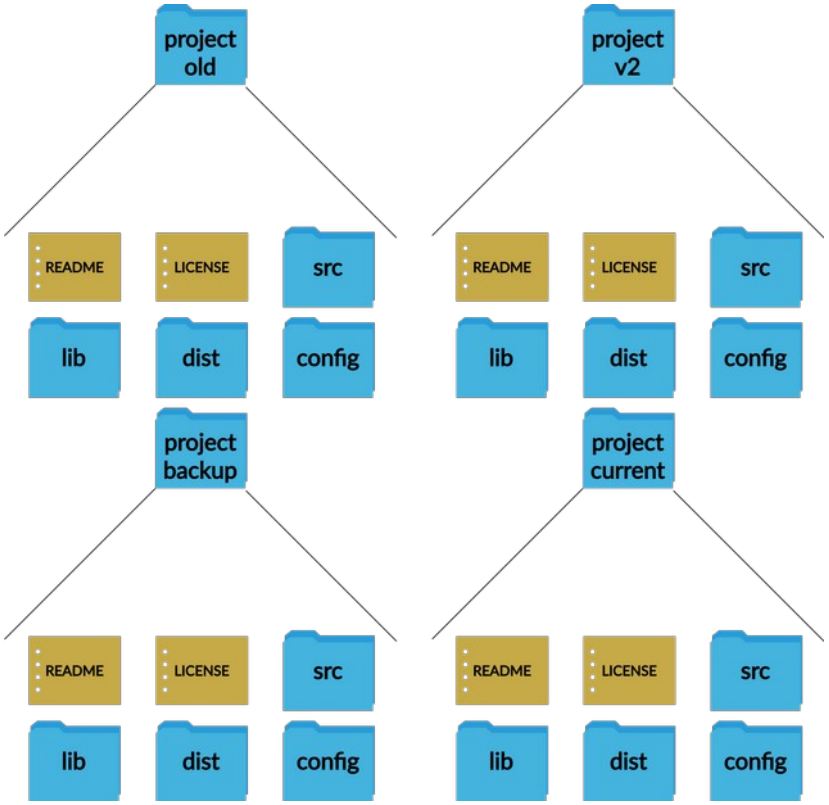
- What happens if
  - You delete a file?



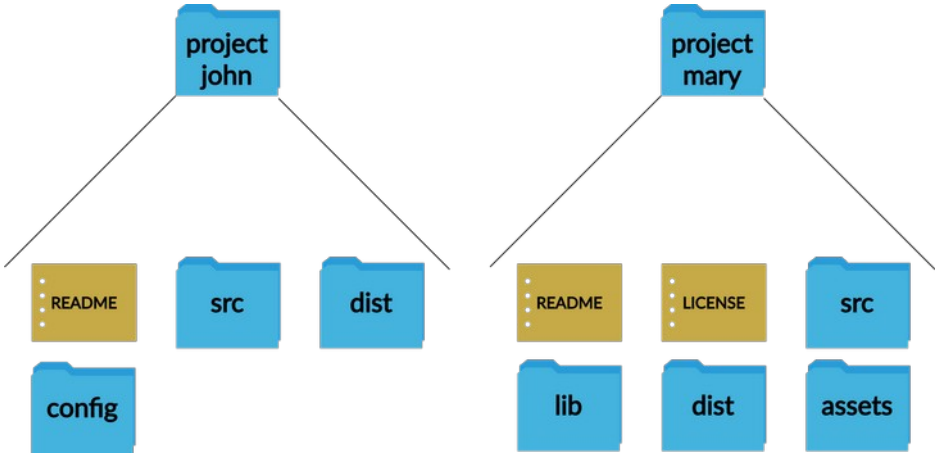
- What happens if
  - You delete a file?
  - You delete the whole folder?



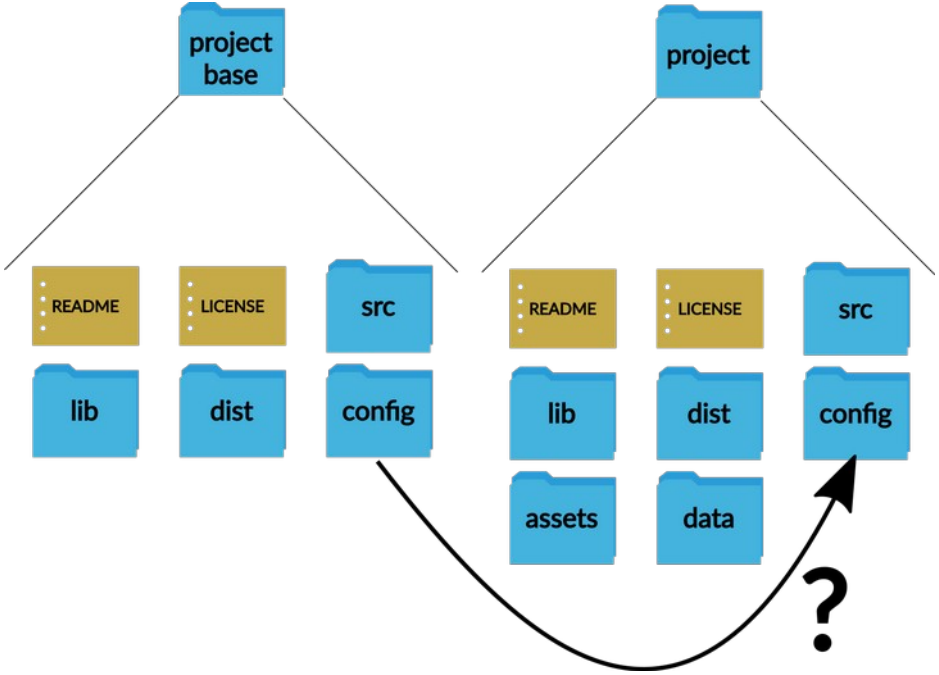
- What happens if
  - You delete a file?
  - You delete the whole folder?
  - You backup the folder?



- What happens if
  - You delete a file?
  - You delete the whole folder?
  - You backup the folder?
  - A team works on the same project?



- What happens if
  - You delete a file?
  - You delete the whole folder?
  - You backup the folder?
  - A team works on the same project?
  - You adapt a template project but want to apply updates?



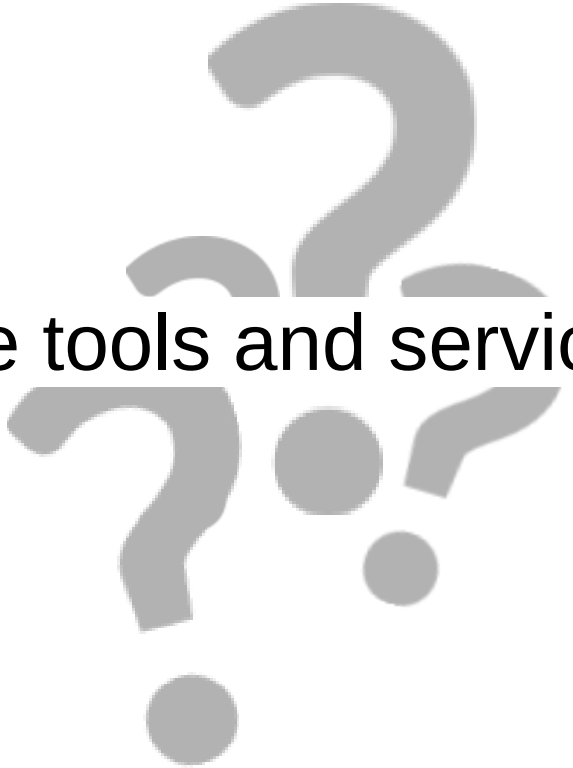
- What happens if
  - You want to backup a team project that adapts an existing project, and then incorporate updates to the base project while simultaneously developing new features

- What happens if
  - You want to backup a team project that adapts an existing project, and then incorporate updates to the base project while simultaneously developing new features
  - **AND** have a central server that regularly runs tests on everyone's contributions

- What happens if
  - You want to backup a team project that adapts an existing project, and then incorporate updates to the base project while simultaneously developing new features
  - **AND** have a central server that regularly runs tests on everyone's contributions
  - **AND** automatically build and deploy a new version of your product every time the main version is updated

- What happens if
  - You want to backup a team project that adapts an existing project, and then incorporate updates to the base project while simultaneously developing new features
  - **AND** have a central server that regularly runs tests on everyone's contributions
  - **AND** automatically build and deploy a new version of your product every time the main version is updated
  - **AND** then split your project into multiple libraries that can be reused across multiple projects while keeping them updated and automatically running tests and automatically publishing them

- What happens if
  - You want to backup a team project that adapts an existing project, and then incorporate updates to the base project while simultaneously developing new features
  - tests on everyone's contributions
  - **AND** automatically build and deploy a new version of your product every time the main version is updated
  - **AND** then split your project into multiple libraries that can be reused across multiple projects while keeping them updated and automatically running tests and automatically publishing them



And do all this with freely available tools and services

- What happens if
  - You want to **backup a team project** that adapts an existing project, and then incorporate updates to the base project while simultaneously developing new features
  - **AND** have a central server that regularly runs tests on everyone's contributions
  - **AND** automatically build and deploy a new version of your product every time the main version is updated
  - **AND** then split your project into multiple libraries that can be reused across multiple projects while keeping them updated and automatically running tests and automatically publishing them

- You use git to
  - create a remote repository which each team member clones

- What happens if
  - You want to backup a team project that **adapts an existing project,** and then incorporate updates to the base project while simultaneously developing new features
  - **AND** have a central server that regularly runs tests on everyone's contributions
  - **AND** automatically build and deploy a new version of your product every time the main version is updated
  - **AND** then split your project into multiple libraries that can be reused across multiple projects while keeping them updated and automatically running tests and automatically publishing them

- You use git to
  - create a remote repository which each team member clones
  - which forks another repository

- What happens if
  - You want to backup a team project that adapts an existing project, and then **incorporate updates to the base project** while simultaneously developing new features
  - **AND** have a central server that regularly runs tests on everyone's contributions
  - **AND** automatically build and deploy a new version of your product every time the main version is updated
  - **AND** then split your project into multiple libraries that can be reused across multiple projects while keeping them updated and automatically running tests and automatically publishing them

- You use git to
  - create a remote repository which each team member clones
  - which forks another repository
  - that you keep as a remote and merge changes from

- What happens if
  - You want to backup a team project that adapts an existing project, and then incorporate updates to the base project while **simultaneously developing new features**
  - **AND** have a central server that regularly runs tests on everyone's contributions
  - **AND** automatically build and deploy a new version of your product every time the main version is updated
  - **AND** then split your project into multiple libraries that can be reused across multiple projects while keeping them updated and automatically running tests and automatically publishing them

- You use git to
  - create a remote repository which each team member clones
  - which forks another repository
  - that you keep as a remote and merge changes from
  - Your repository has multiple branches

- What happens if
  - You want to backup a team project that adapts an existing project, and then incorporate updates to the base project while simultaneously developing new features
  - **AND** have a **central server that regularly runs tests on everyone's contributions**
  - **AND** automatically build and deploy a new version of your product every time the main version is updated
  - **AND** then split your project into multiple libraries that can be reused across multiple projects while keeping them updated and automatically running tests and automatically publishing them

- You use git to
  - create a remote repository which each team member clones
  - which forks another repository
  - that you keep as a remote and merge changes from
  - Your repository has multiple branches
  - You add a Continuous Integration tool to your git server that automatically builds and tests the code on push
  -

- What happens if
  - You want to backup a team project that adapts an existing project, and then incorporate updates to the base project while simultaneously developing new features
  - **AND** have a central server that regularly runs tests on everyone's contributions
  - **AND** automatically build and **deploy a new version of your product** every time the main version is updated
  - **AND** then split your project into multiple libraries that can be reused across multiple projects while keeping them updated and automatically running tests and automatically publishing them

- You use git to
  - create a remote repository which each team member clones
  - which forks another repository
  - that you keep as a remote and merge changes from
  - Your repository has multiple branches
  - You add a Continuous Integration tool to your git server that automatically builds and tests the code on push
  - You get your CI tool fire a webhook to a service that builds and deploys your code

- What happens if
  - You want to backup a team project that adapts an existing project, and then incorporate updates to the base project while simultaneously developing new features
  - **AND** have a central server that regularly runs tests on everyone's contributions
  - **AND** automatically build and deploy a new version of your product every time the main version is updated
  - **AND** then **split your project into multiple libraries that can be reused across multiple projects while keeping them updated** and automatically running tests and automatically publishing them

- You use git to
  - create a remote repository which each team member clones
  - which forks another repository
  - that you keep as a remote and merge changes from
  - Your repository has multiple branches
  - You add a Continuous Integration tool to your git server that automatically builds and tests the code on push
  - You get your CI tool fire a webhook to a service that builds and deploys your code
  - You turn your libraries into submodules

- What happens if
  - You want to backup a team project that adapts an existing project, and then incorporate updates to the base project while simultaneously developing new features
  - **AND** have a central server that regularly runs tests on everyone's contributions
  - **AND** automatically build and deploy a new version of your product every time the main version is updated
  - **AND** then split your project into multiple libraries that can be reused across multiple projects while keeping them updated and **automatically running tests and automatically publishing them**

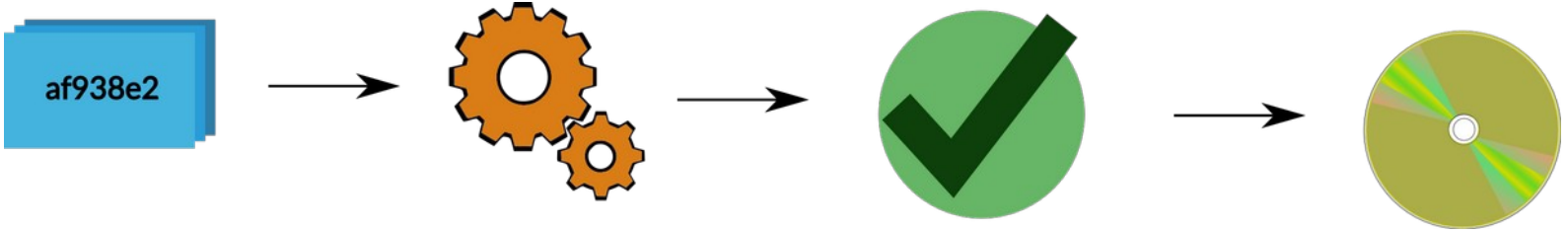
- You use git to
  - create a remote repository which each team member clones
  - which forks another repository
  - that you keep as a remote and merge changes from
  - Your repository has multiple branches
  - You add a Continuous Integration tool to your git server that automatically builds and tests the code on push
  - You get your CI tool fire a webhook to a service that builds and deploys your code
  - You turn your libraries into submodules
  - That integrate with CI tools and automated publishing workflows

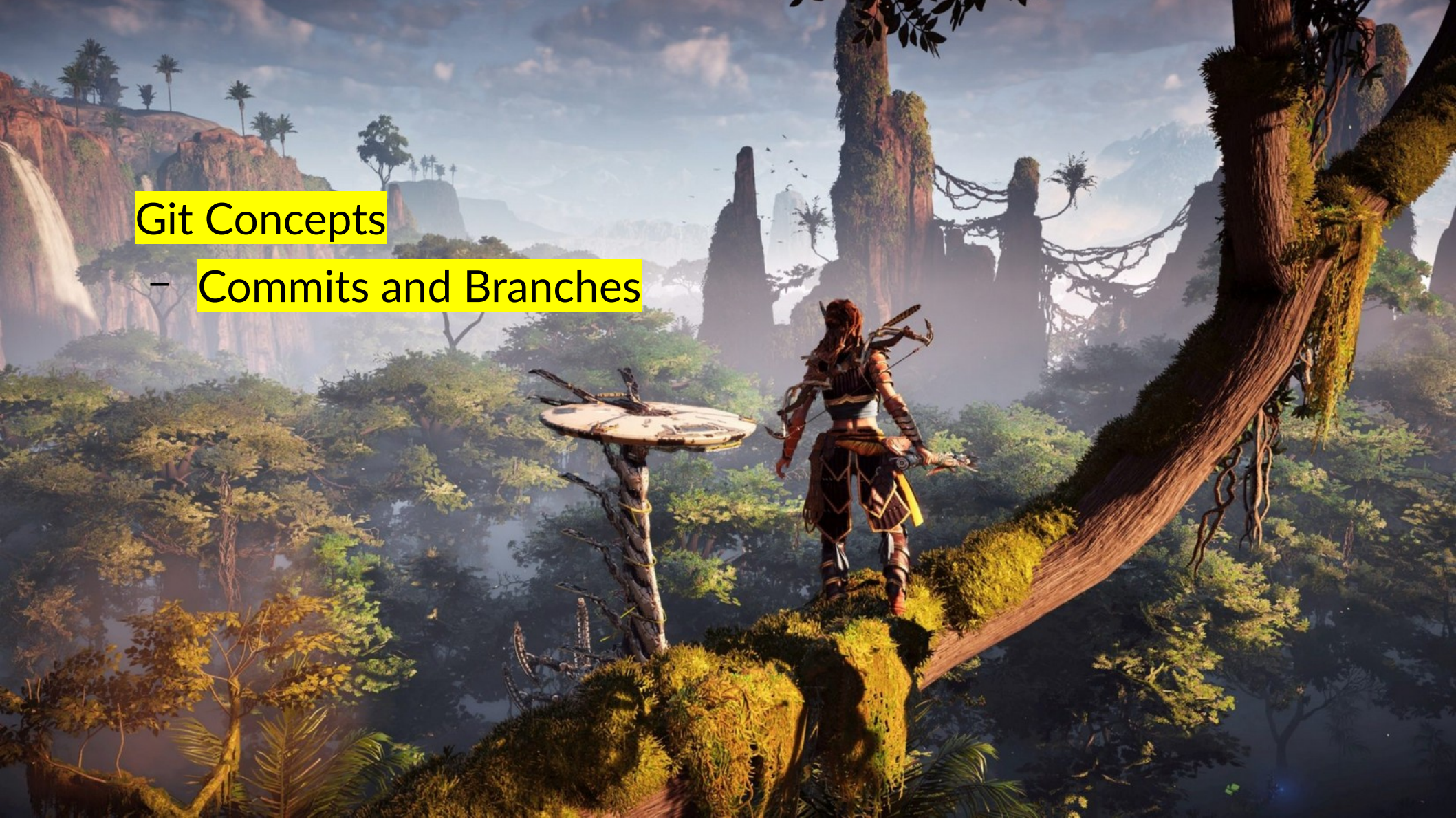
Workflow

- Workflow is important
  - Your time is important
  - Your enthusiasm is important
- Bad workflow leads to bad projects
  - Poor code quality
  - Cutting corners
  - Slow updates
  - Integration hell

- Good workflows make development easier

- Continuous Delivery

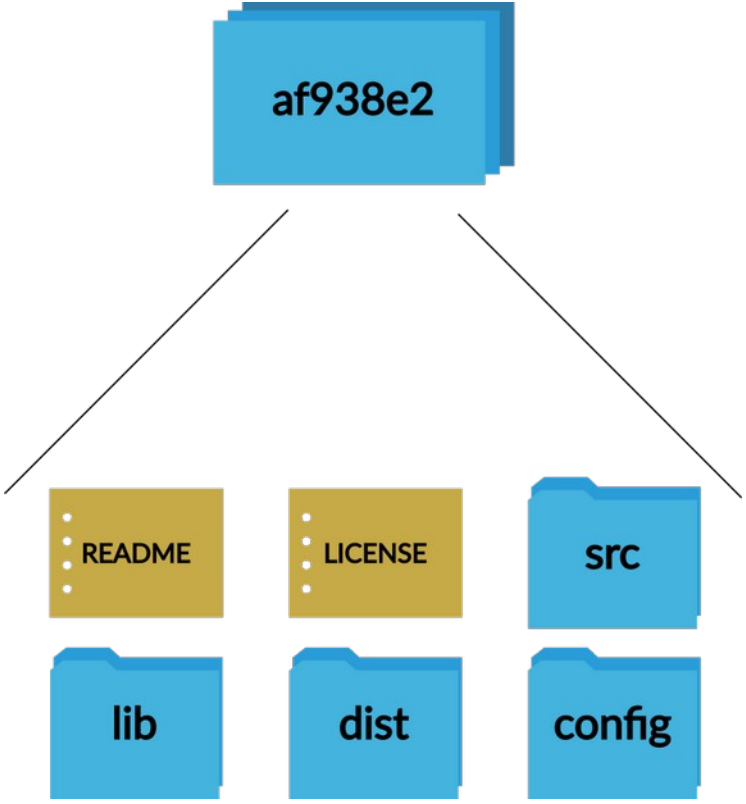




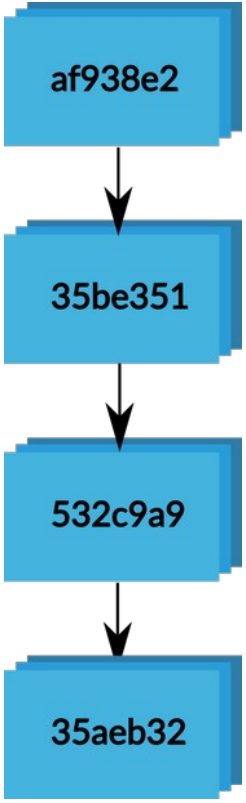
Git Concepts

- Commits and Branches

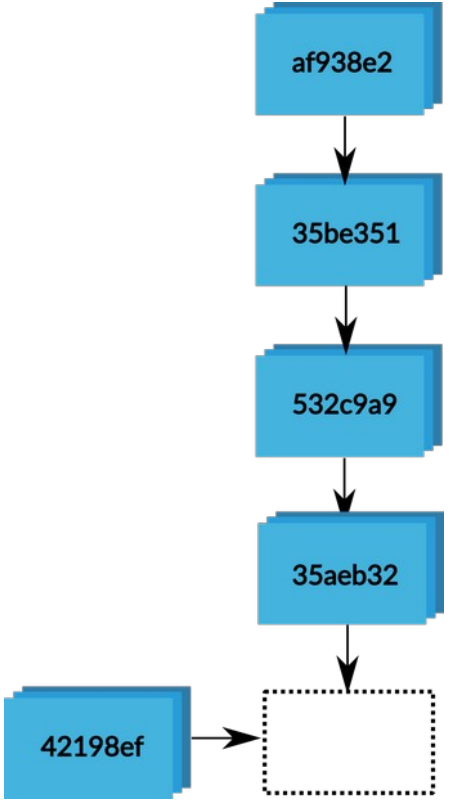
- Commit (noun)
  - A snapshot of your project



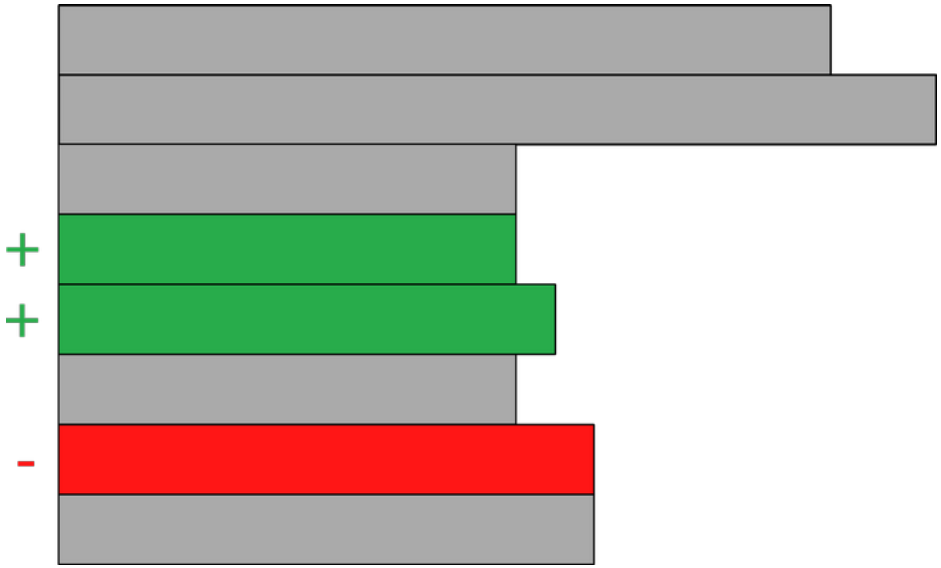
- Branch (noun)
  - A sequence of commits
  - Shows how your project has changed over time



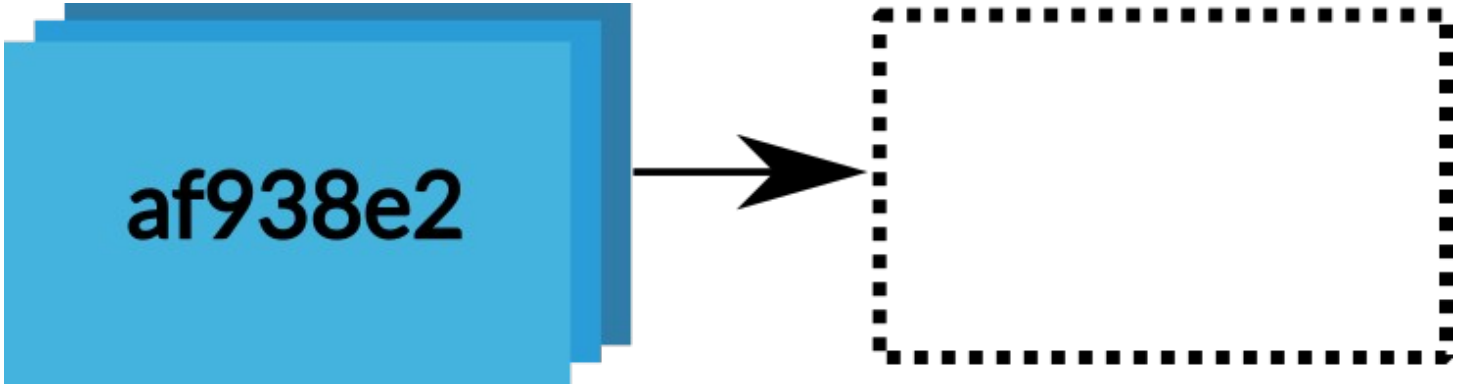
- Commit (verb)
  - Take a snapshot of your project and add it onto the end of the current branch



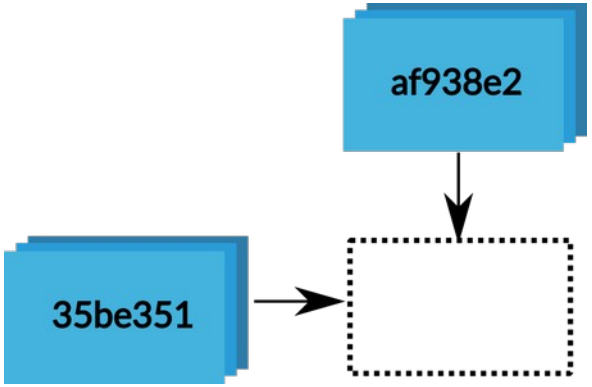
- Commits
  - Commits are calculated line-by-line
  - A commit stores just the changes that have been made since the last commit



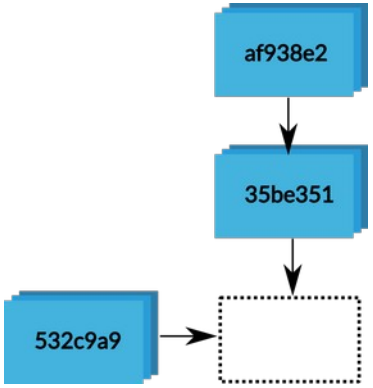
- Simple workflow example
  - Create a new repository and make an initial commit



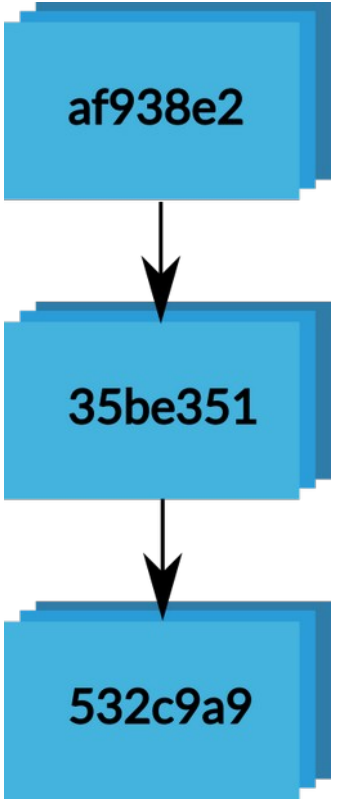
- Simple workflow example
  - Create a new repository and make an initial commit
  - Do some work, then commit it



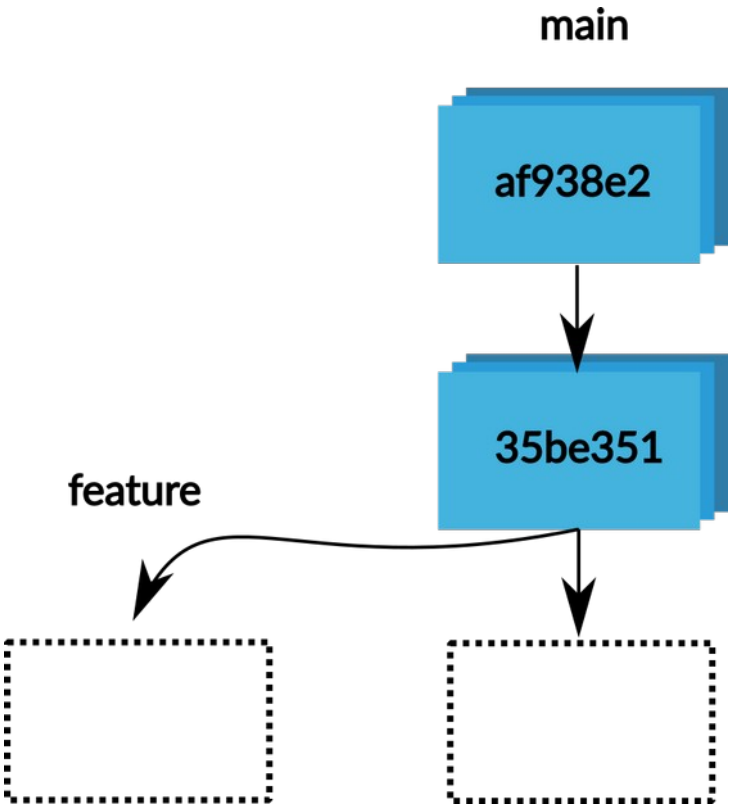
- Simple workflow example
  - Create a new repository and make an initial commit
  - Do some work, then commit it
  - Do some more work, then commit it



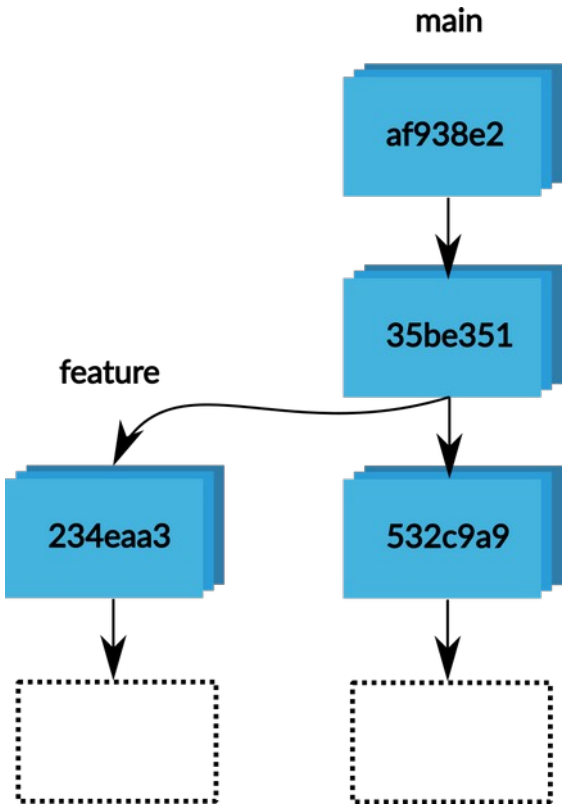
- Simple workflow example
  - Now you’ve got a branch with
    - The current state of your project
    - All changes made to your project



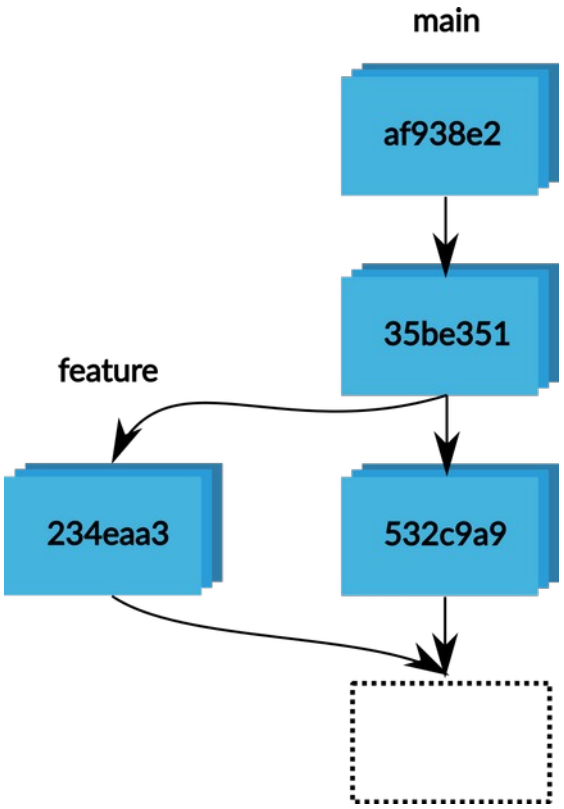
- Branch (verb)
  - Take a branch and split a new one off it
  - Now you’ve got two branches



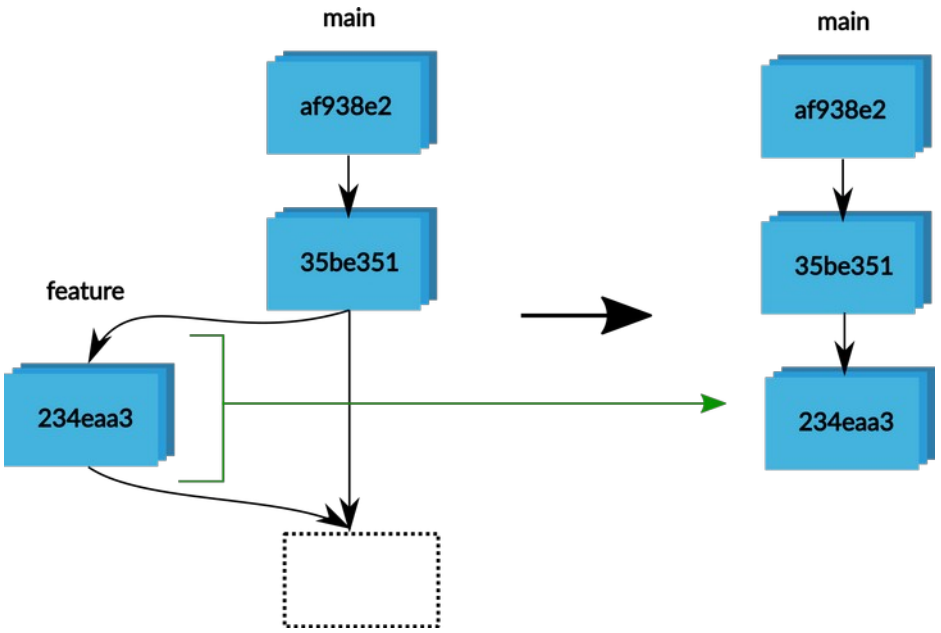
- Multiple branches
  - You can add commits to each branch
  - This gives you two different versions of your project



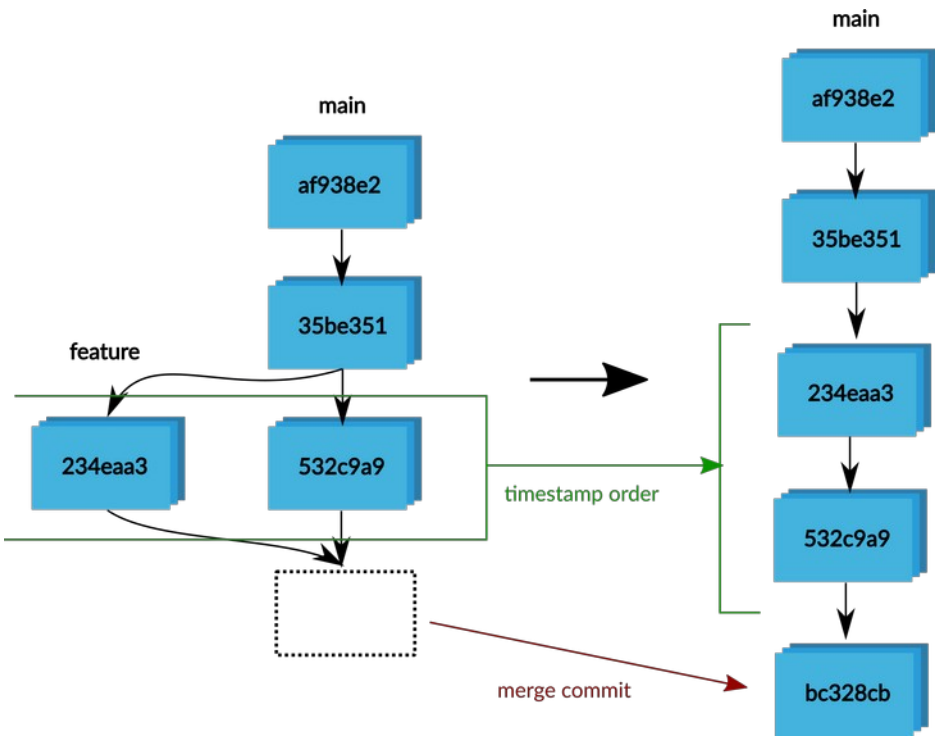
- Merge
  - Combine the changes made in two branches



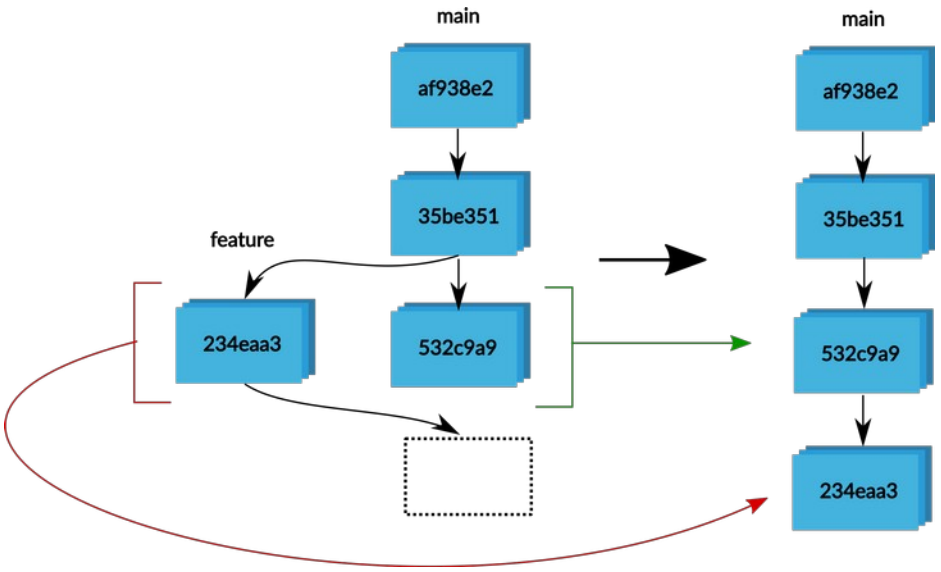
- Merge (Fast Forward)
  - There are only commits on the branch to merge in
  - These commits can be added on to the end of the main branch



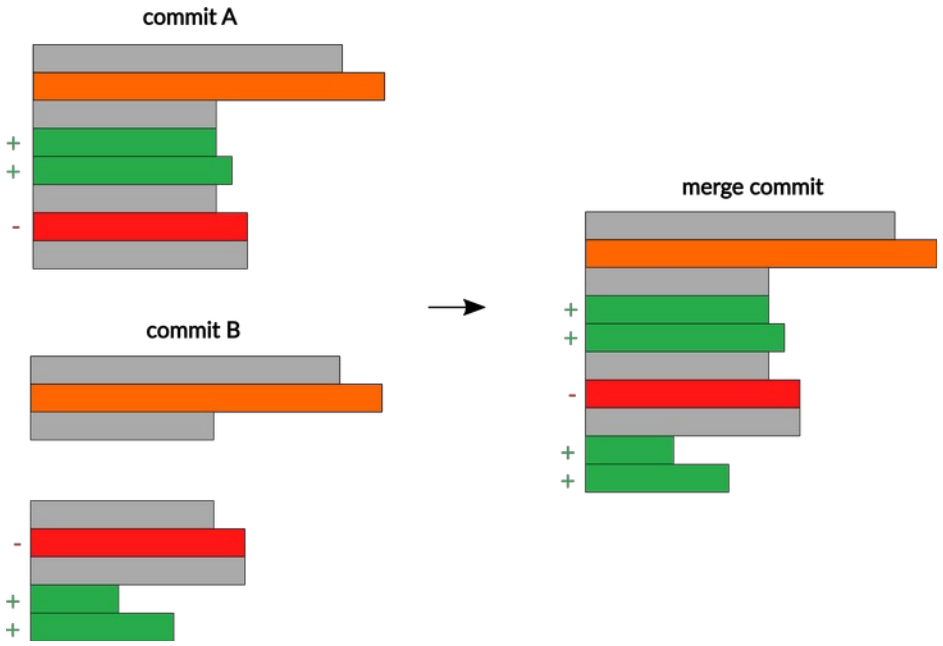
- Merge (Standard Merge)
  - Adds commits in timestamp order
  - Create a new commit for the merge



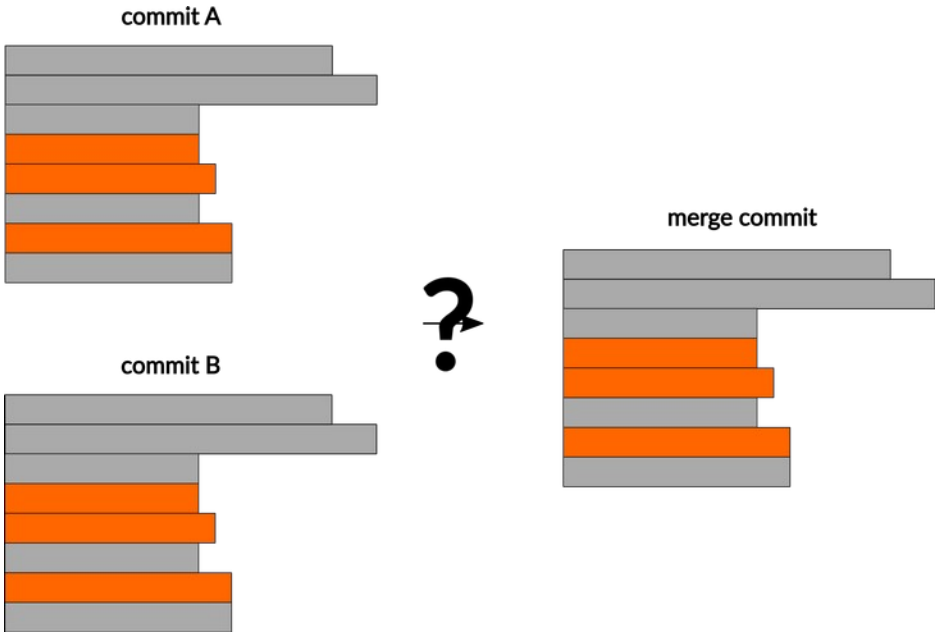
- Merge (Rebase)
  - Replay the feature branch on top of the new commits on the main branch



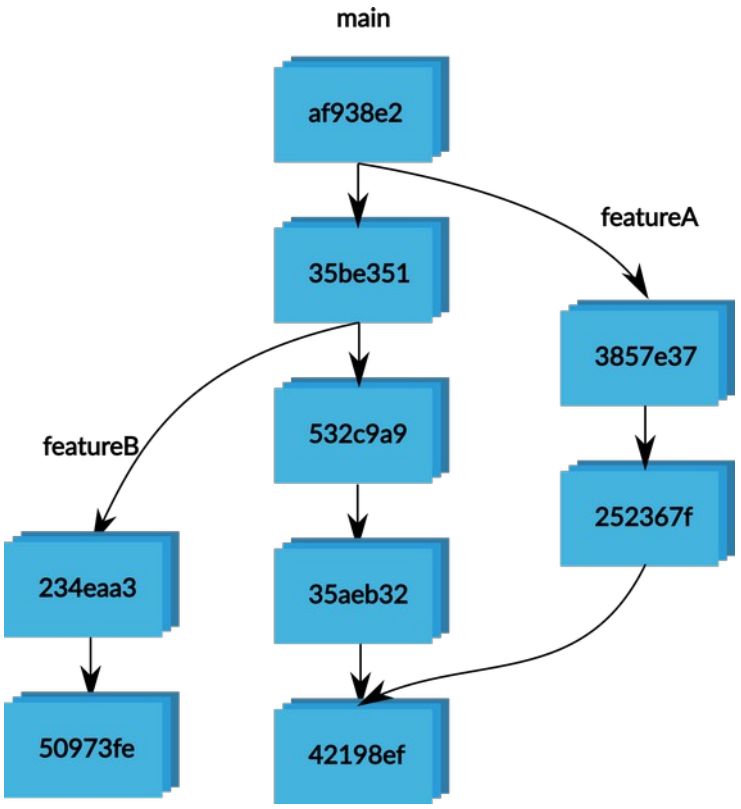
- Resolving a merge
  - Git merges line-by-line
  - If the branches change different lines everything resolves automatically



- Resolving a merge
  - If both branches change the same line, you need to resolve the merge manually
  - Git will edit the files with the options. You delete the appropriate section and commit the merge



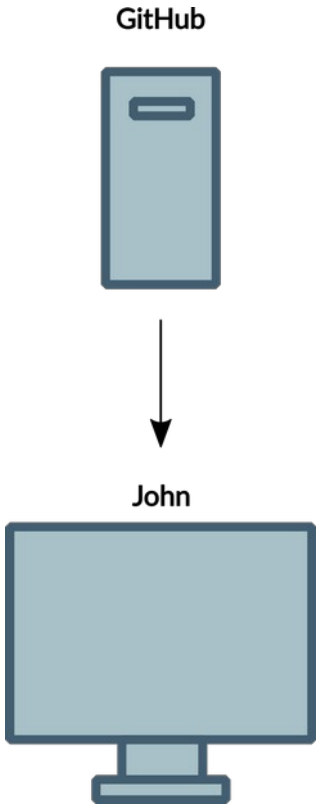
- Repository (noun)
  - Everything git keeps track of for your project
  - A collection of branches



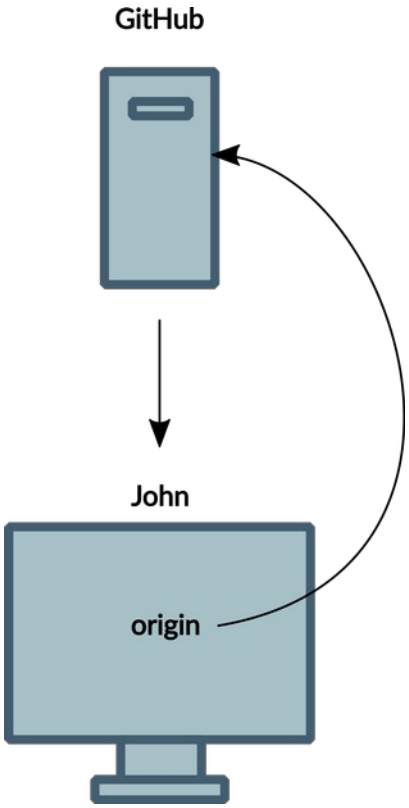
Git Concepts

Working with remotes

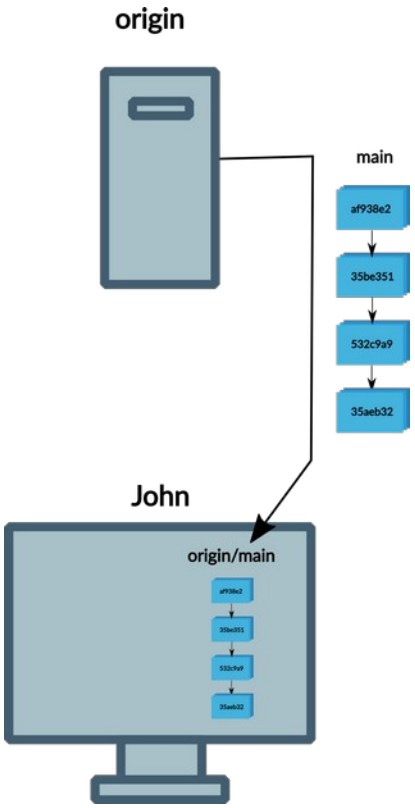
- Clone (verb)
  - Make a copy of a repository held on a git server



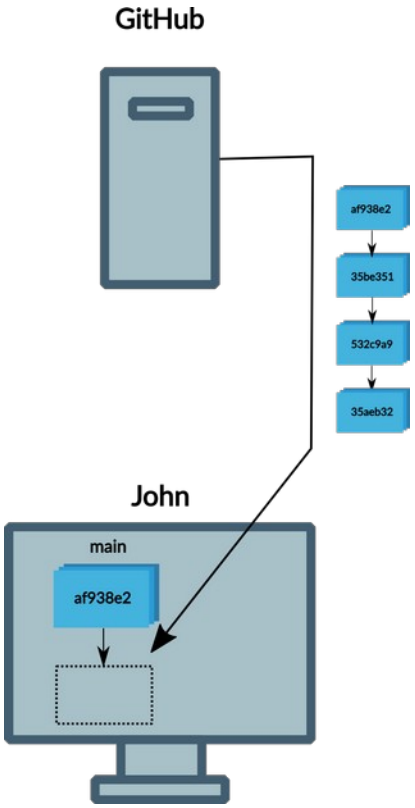
- Remote (noun)
  - A reference in your git repo to a repository elsewhere
  - When you clone a repo, it creates a remote called origin



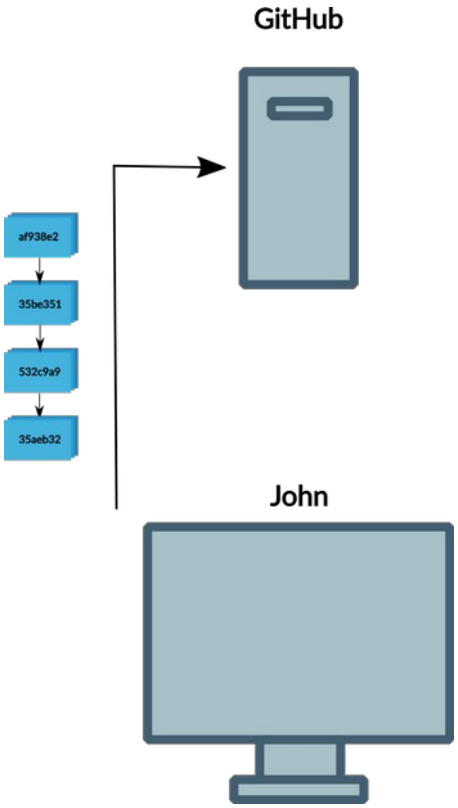
- Fetch (verb)
  - Download commits made on a remote branch
  - Fetching branch main from origin would put result in branch origin/main



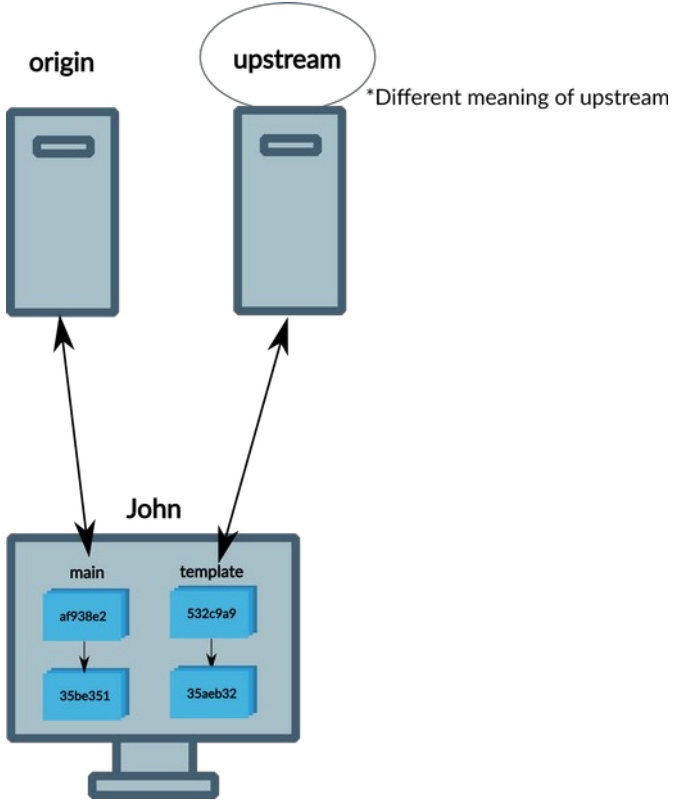
- Pull (verb)
  - Fetch **and merge** commits made on a remote branch



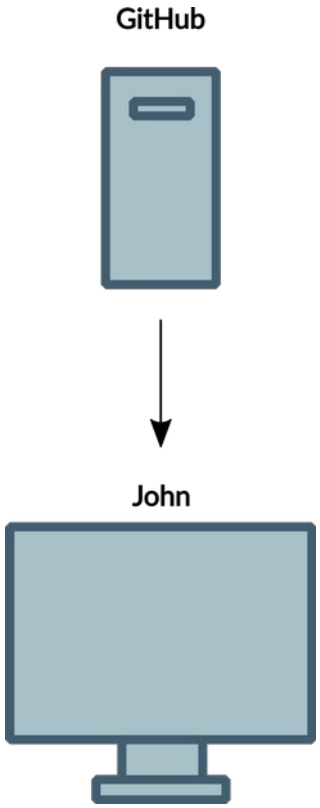
- Push (verb)
  - Upload new commits in the local repository to a remote branch



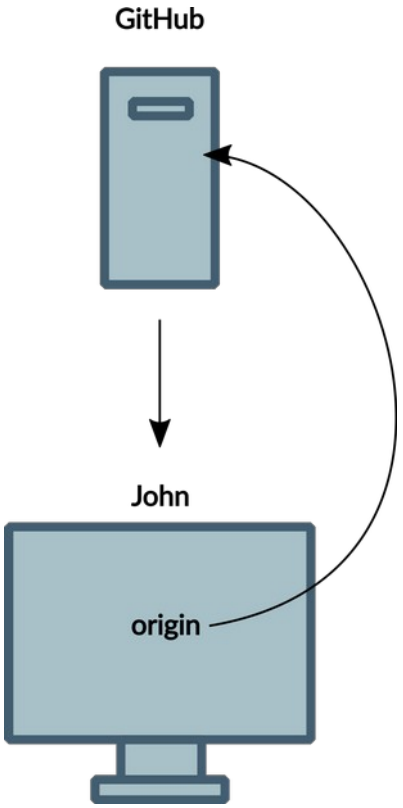
- Upstream (noun)
  - Which remote branch our branch should fetch/pull from and push to
  - Different branches may have upstreams on different remotes



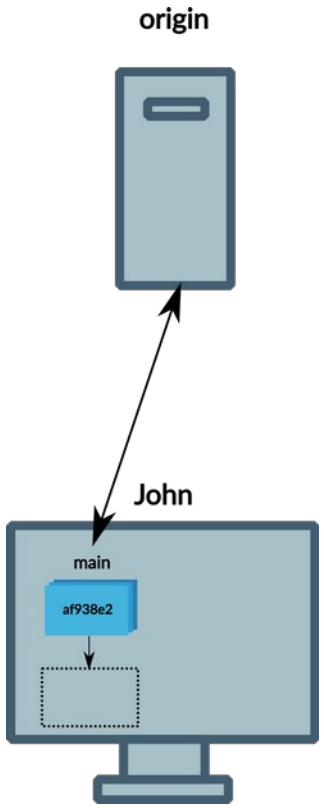
- Example workflow
  - Clone a repository



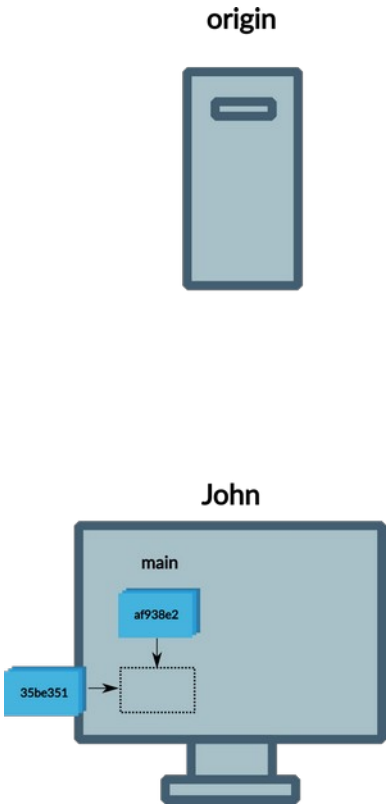
- Example workflow
  - Clone a repository
    - Remote **origin** is set automatically



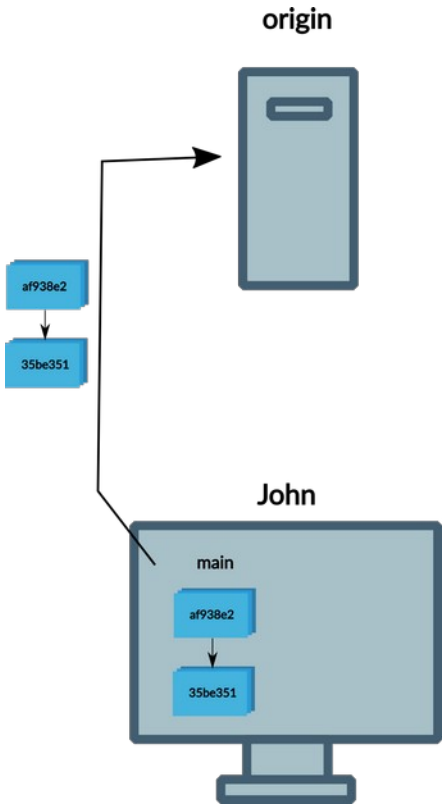
- Example workflow
  - Clone a repository
    - Remote **origin** is set automatically
  - Set upstream for branch **main**



- Example workflow
  - Clone a repository
    - Remote **origin** is set automatically
  - Set upstream for branch **main**
  - Commit changes on branch **main**



- Example workflow
  - Clone a repository
    - Remote **origin** is set automatically
  - Set upstream for branch **main**
  - Commit changes on branch **main**
  - Push commits to remote branch



- **Question:** What would be the steps involved in the following:
  - Create a repository locally
  - Add content to it
  - Create an empty repository on GitHub
  - Push your content to the empty repository

- **Answer:** What would be the steps involved in the following:
  - Create a repository locally
    - git init
  - Add content to it
    - git commit
  - Create an empty repository on GitHub
    - git remote add
    - git push --set-upstream
  - Push your content to the empty repository
    - git push

- **Question:** What would be the steps involved in the following:
  - Copy an existing repo (upstream)
  - Make a new GitHub repo for it
  - Work on a new feature
  - Update with changes to upstream
  - Incorporate new feature into main branch
  - Push to GitHub repo

- **Answer:** What would be the steps involved in the following:
  - Copy an existing repo (upstream)
    - git clone
  - Make a new GitHub repo for it
    - git remote add
  - Work on a new feature
    - git branch
    - git commit
  - Update with changes to upstream
    - git pull
  - Incorporate new feature into main branch
    - git merge
  - Push to GitHub repo
    - git push --set-upstream

Basic tasks in Git

- Git tools
  - You can use graphical tools with git
  - The instructions here are command line
    - Clear what’s happening
    - Learn once – use anywhere
    - You need to resort to command line to do anything complicated anyway

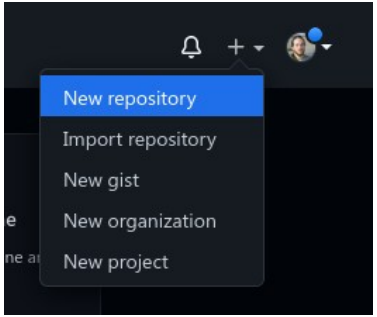
- We will see
  - HTTPS authentication with a Personal Access Token
  - SSH authentication with an SSH Key
  - Create a repository
  - Set a remote and push commits
  - Clone a repository

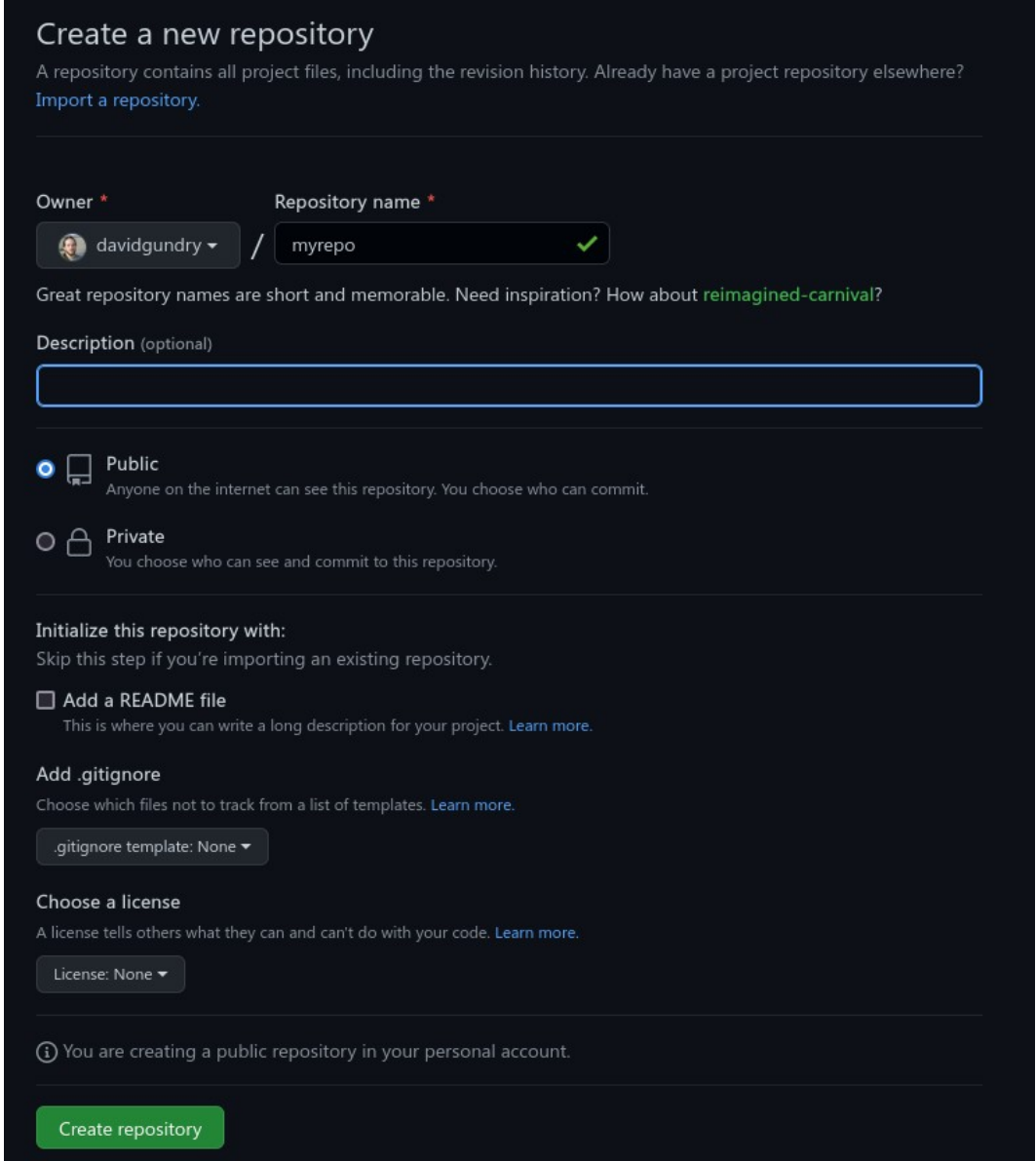
Clone a repository

- Here we're going to see
  - How to clone an existing repository

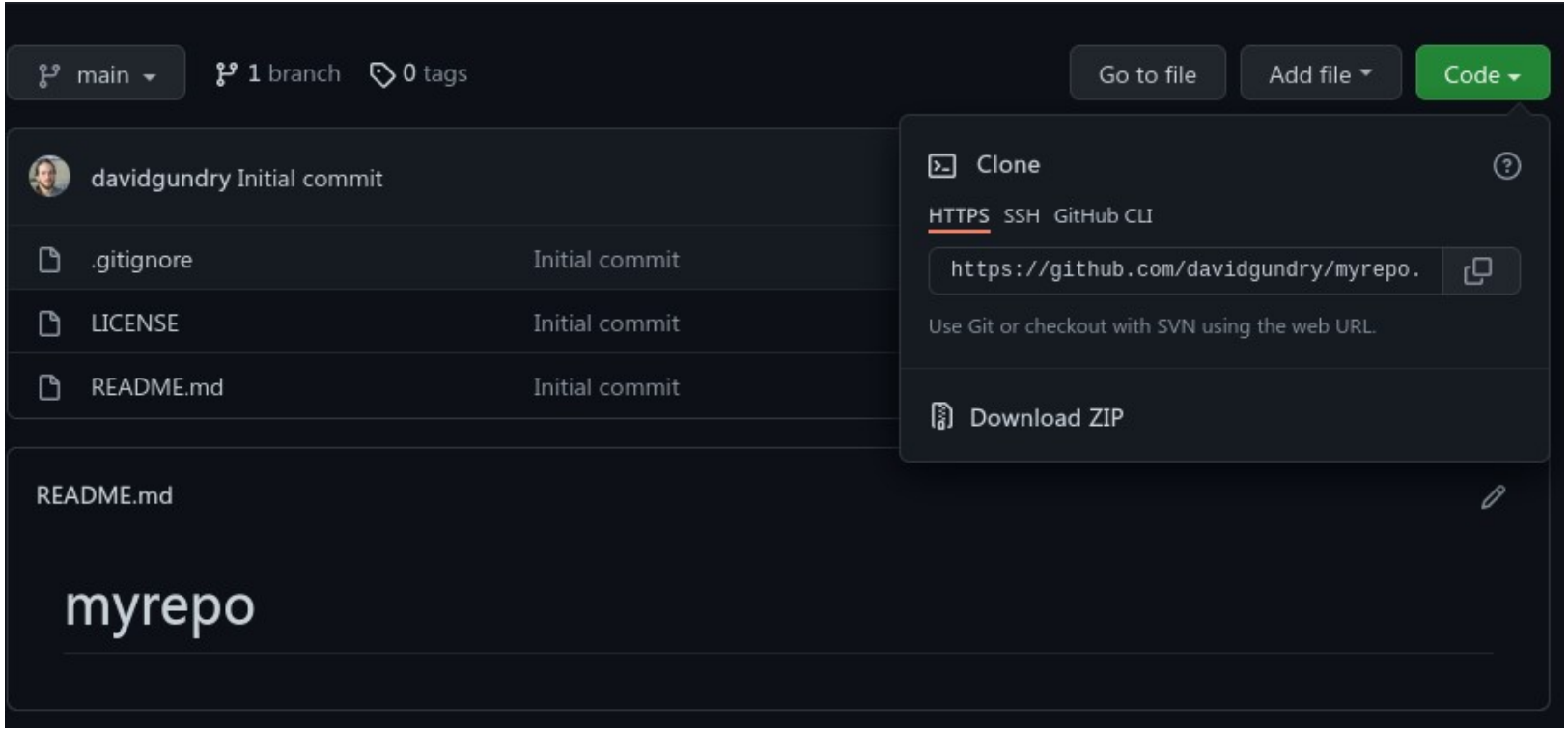
Clone a repository

- Find or create a repository on e.g. GitHub
  - Here I'm assuming there is content in the repository, so feel free to add Readme, .gitignore, and license

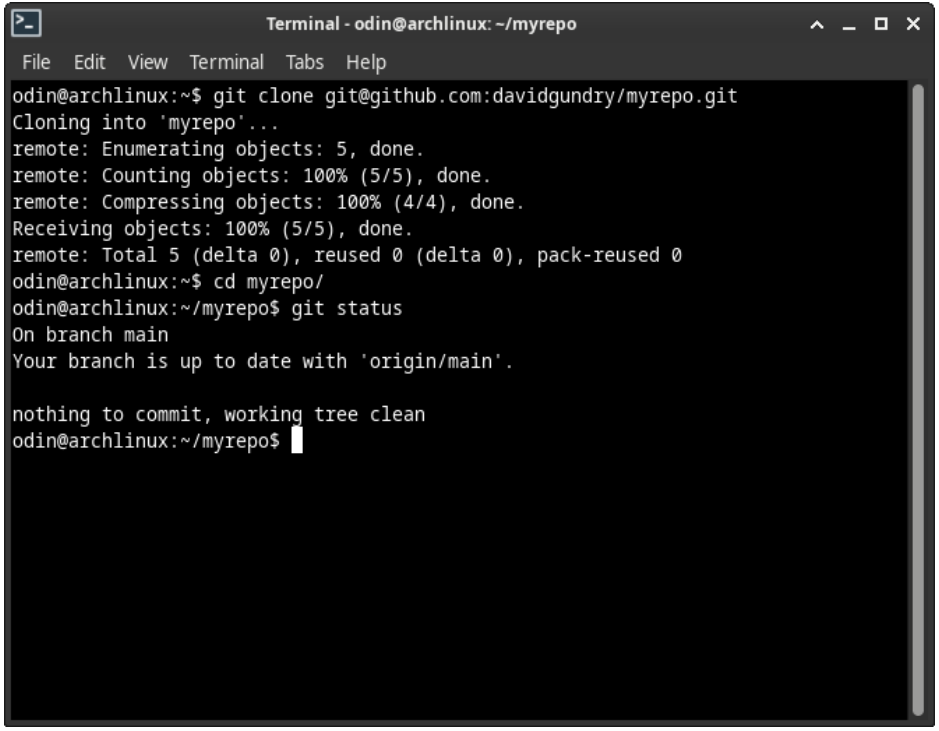




- Find the URL of the repository



- Clone the repository
  - `git clone <repository-url>`
- Enter the directory
  - `cd`
- Check the status of the repository
  - `git status`

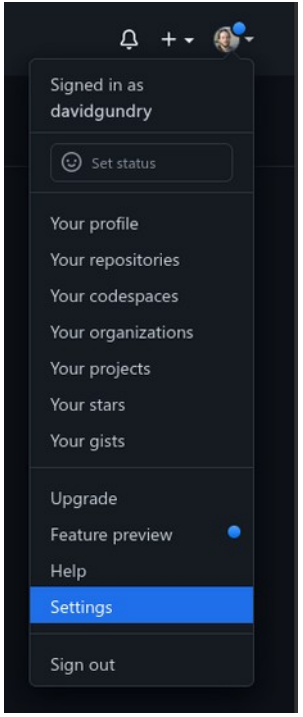


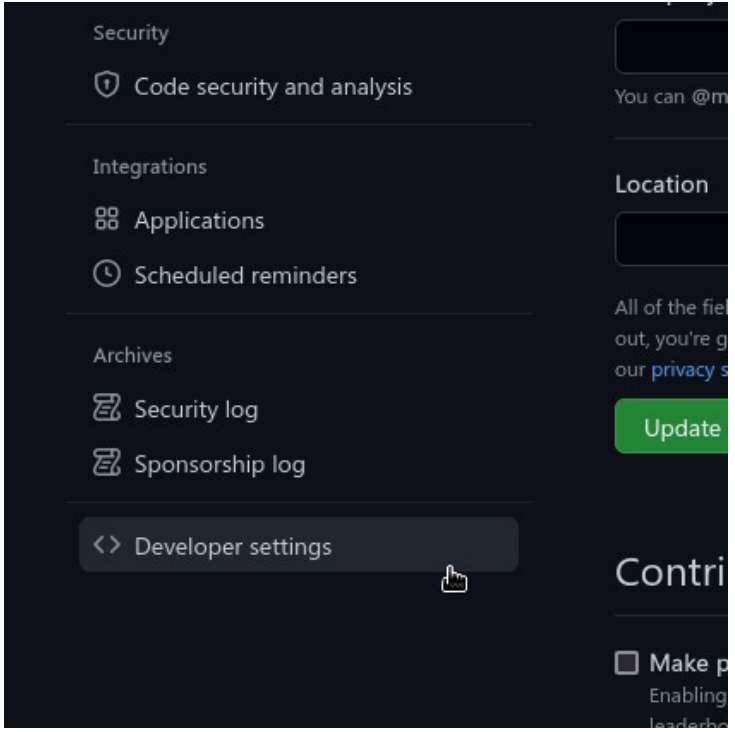
- **Try it now:**
  - Create a public repository on GitHub
  - Clone the repository

HTTPS authentication with Personal Access Token

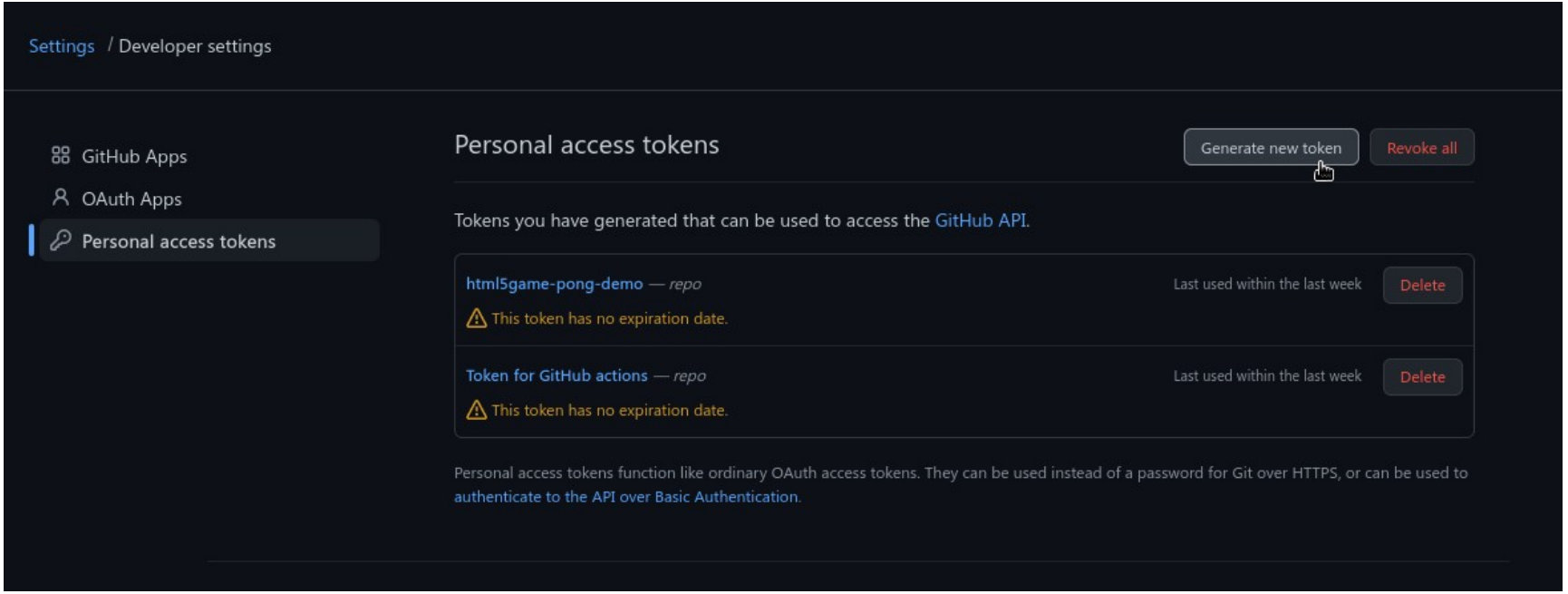
- GitHub doesn’t allow authentication with username and password
- One method is to create a Personal Access Token
  - This is basically your password, so keep it safe

- Go to GitHub Settings > Developer Settings

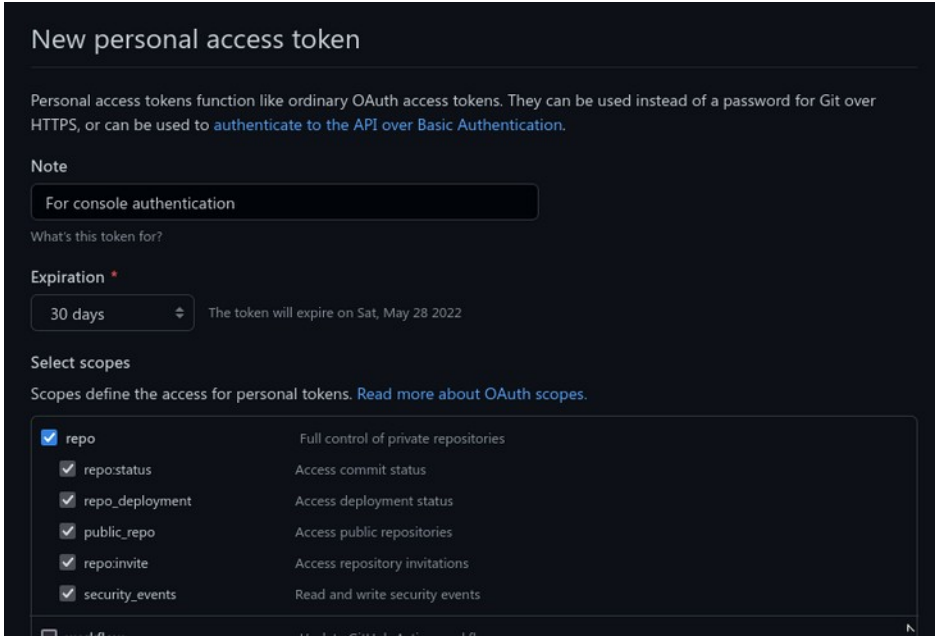




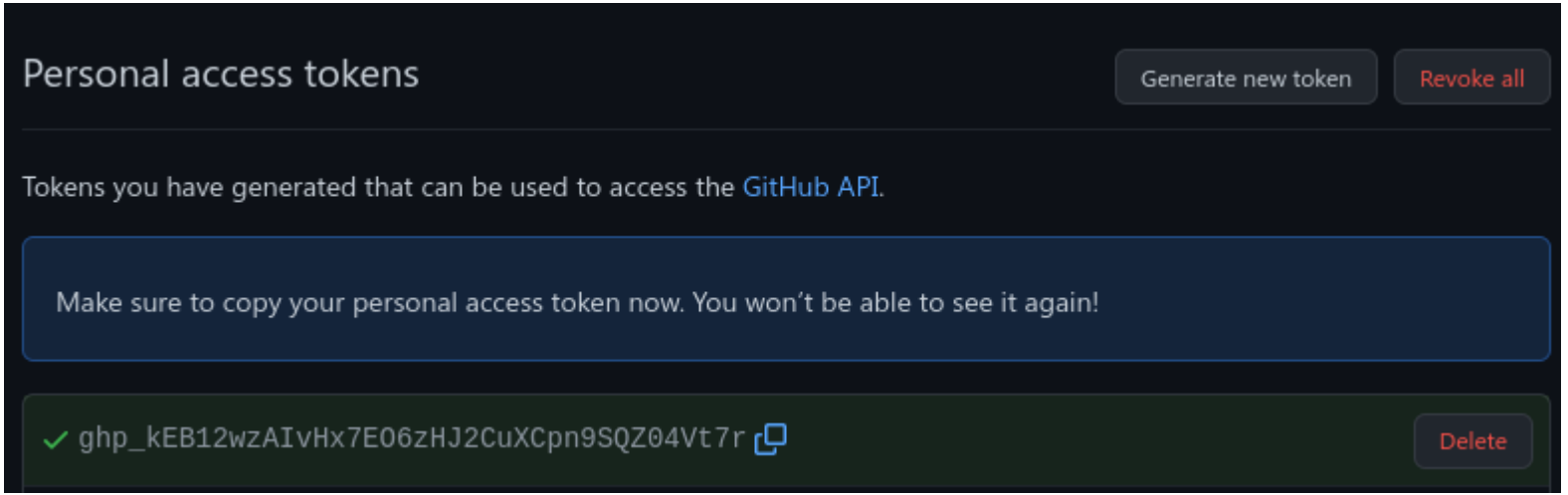
- Click generate new token



- Give the token control over repositories



- Copy the code it gives you
- This is what to use instead of your password in the console

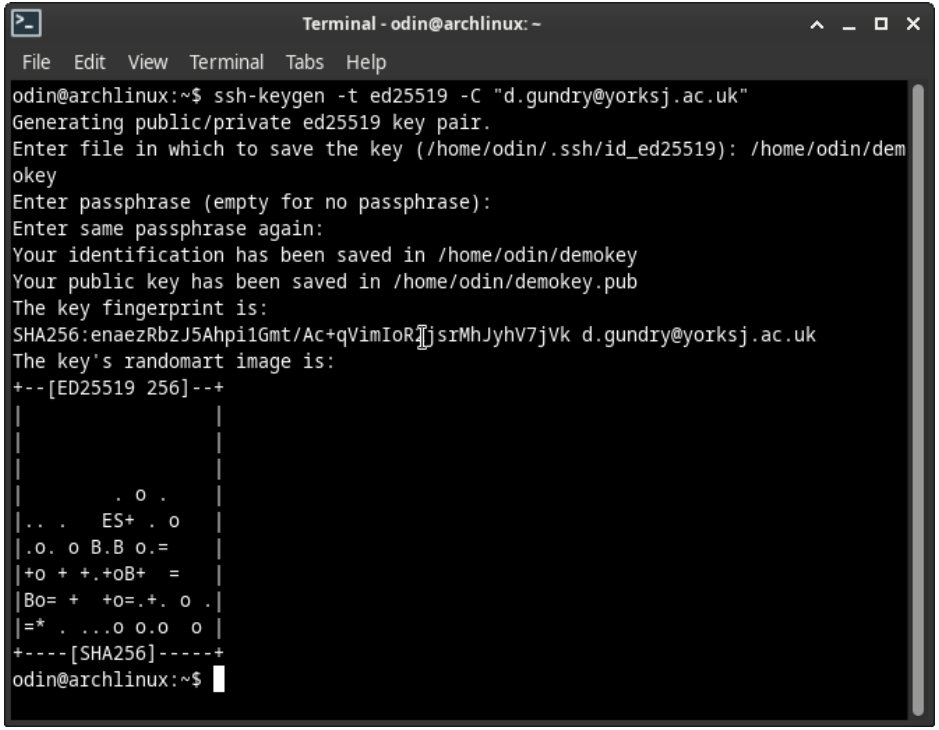


- If you connect to a remote with an HTTPS url you will need to type in your username and enter this token as your password
- **Test it now:**
  - Create a private repository on GitHub
  - Set up a Personal Access Token
  - Clone the repository using the HTTPS url

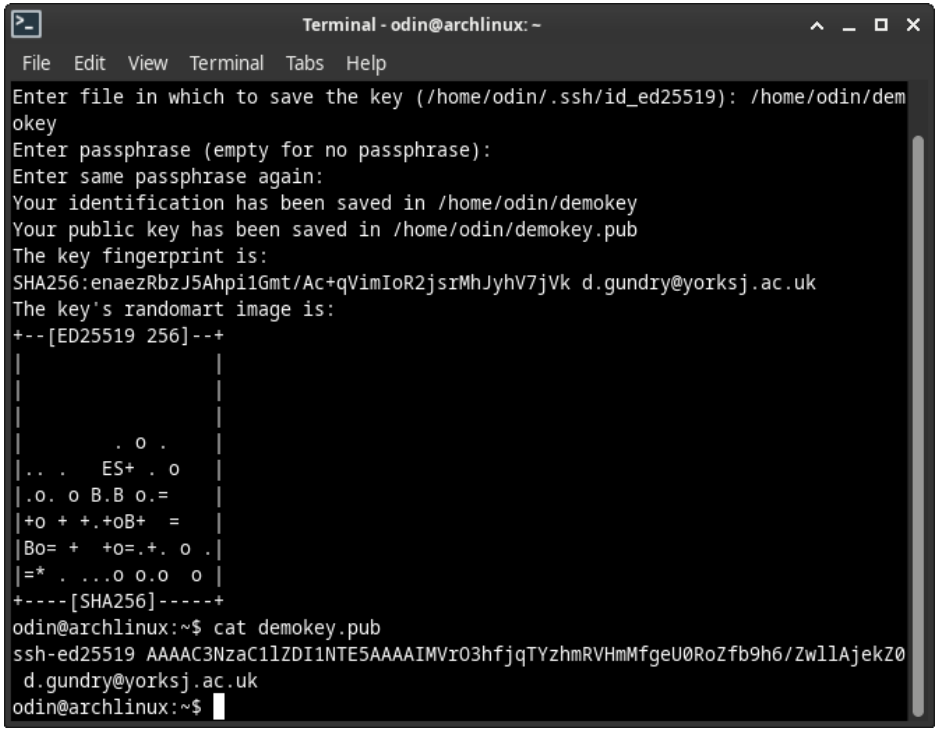
SSH authentication with SSH key

- Another method is to create an ssh key

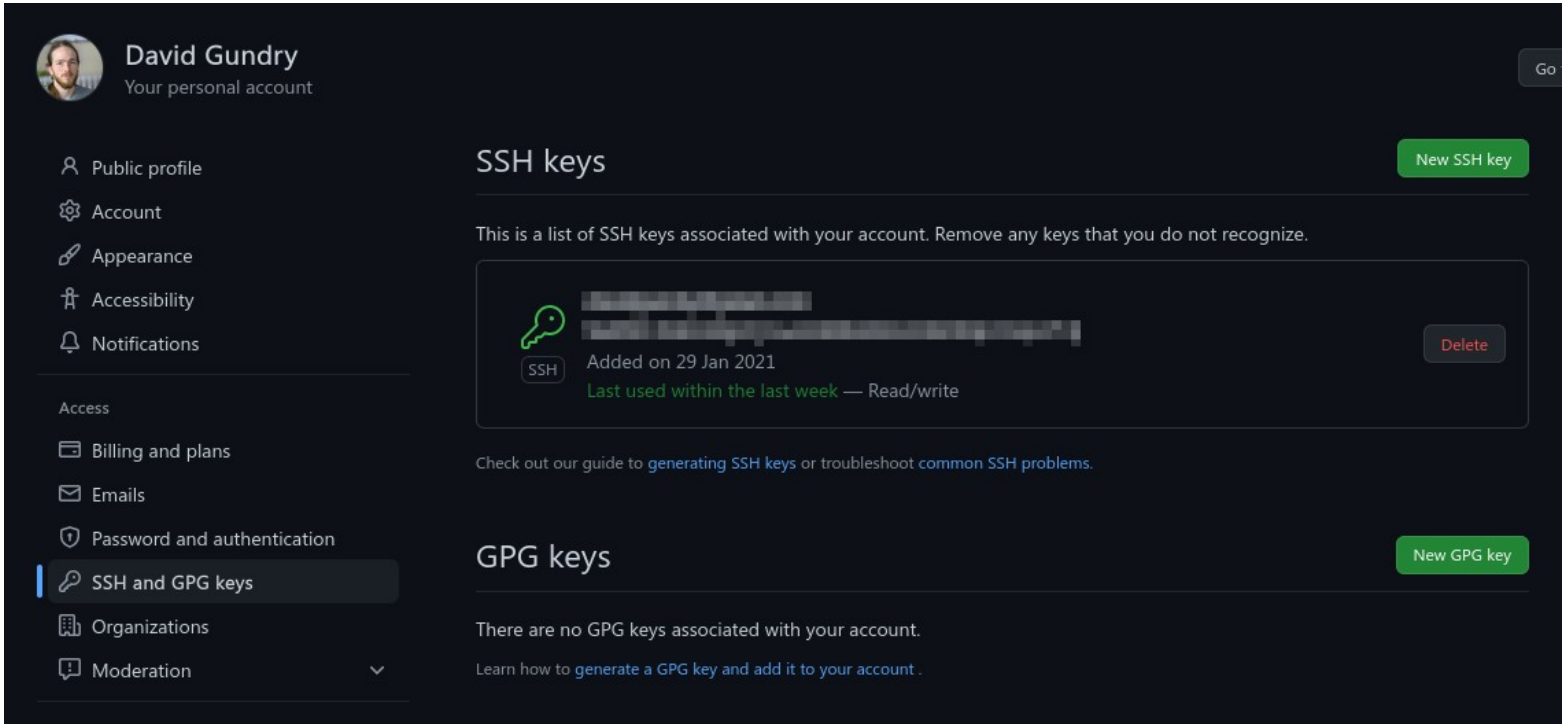
- Generate your ssh key
  - ssh-keygen -t ed25519 -C “<your email>”



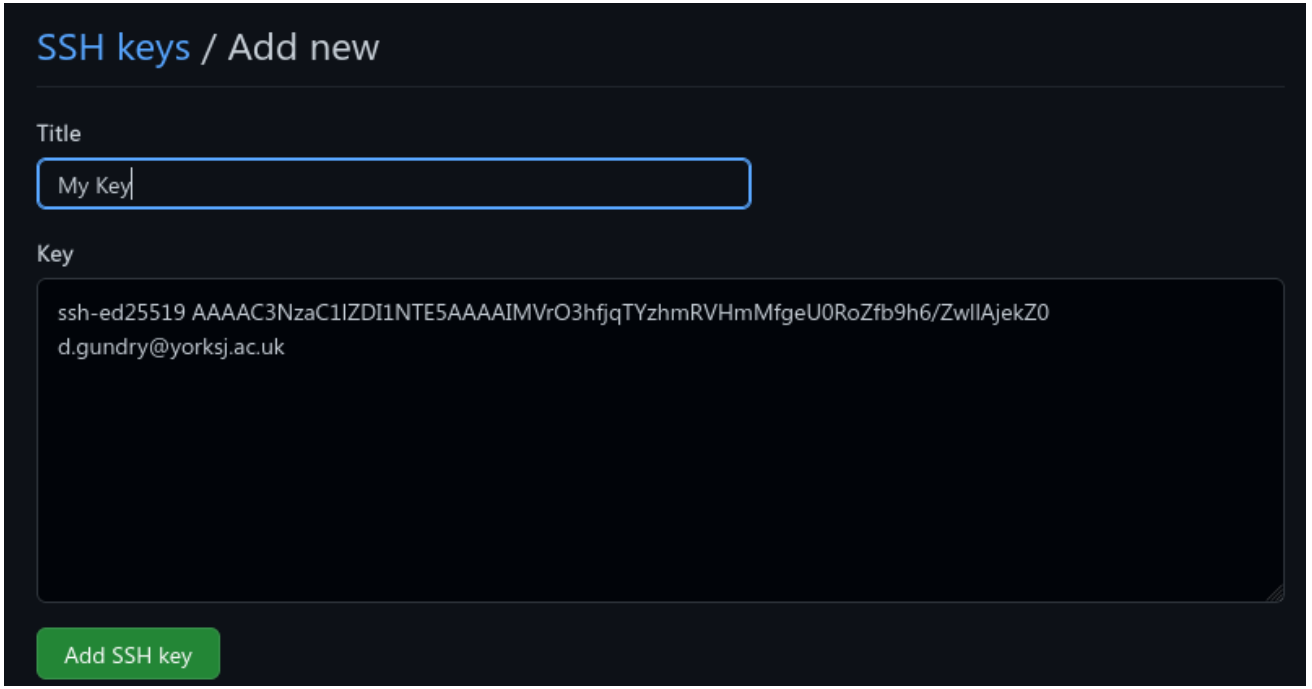
- Copy the content of the .pub file created
  - This is your public key
  - The other file is the private key



- Go to GitHub Settings > SSH and GPG keys > New SSH Key

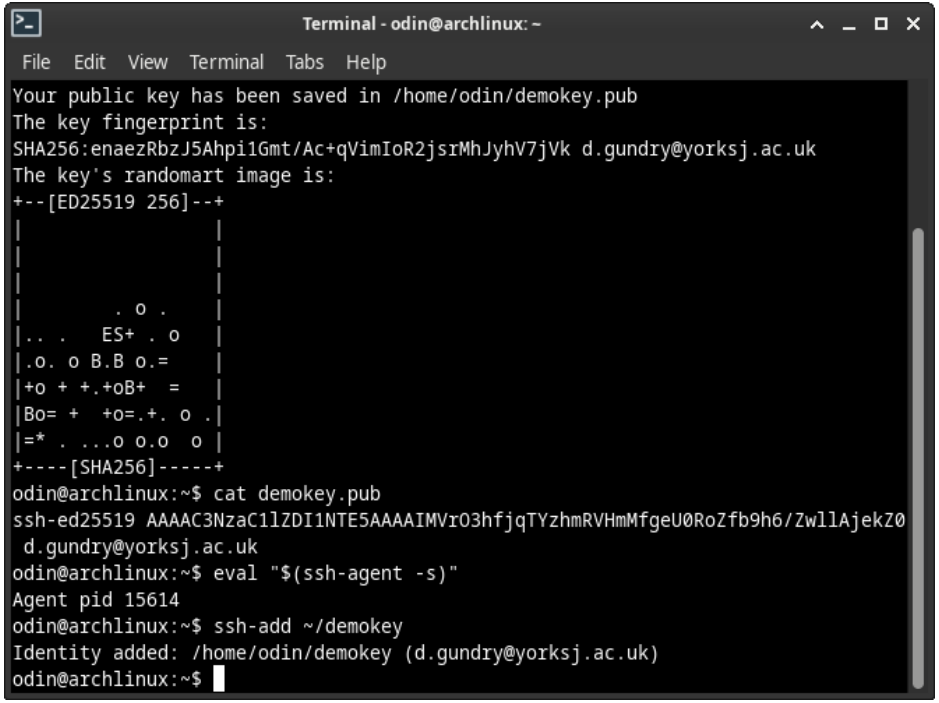


- Paste the .pub file content into the Key field and click Add SSH key



- Before you can authenticate with your key, it needs to be added to ssh-agent
  - Start ssh-agent
  - add your **private** key

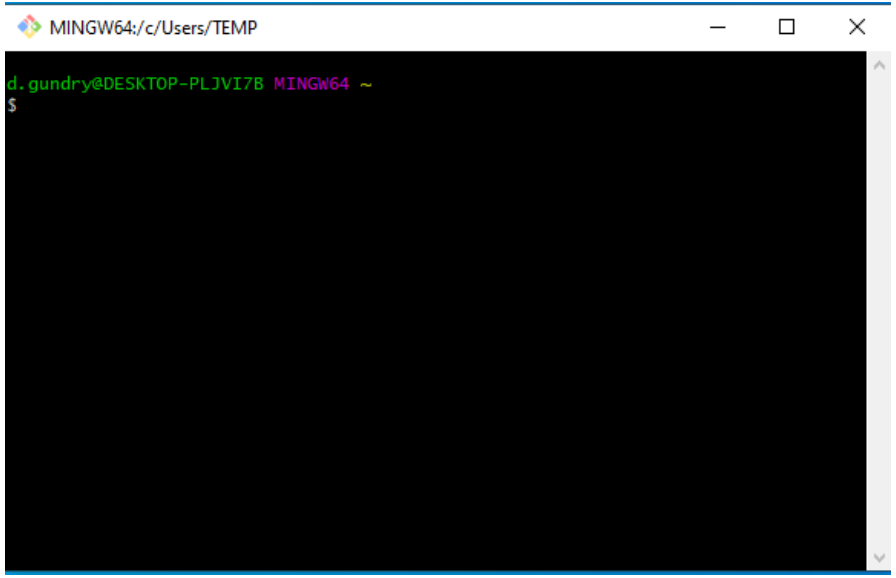
  ssh-add /path/to/key/file
- It will remember your key but only for the session



- If you connect to a remote with an SSH url you will automatically be authenticated
- **Test it now:**
  - Create a private repository on GitHub
  - Set up an SSH key
  - Clone the repository using the SSH url

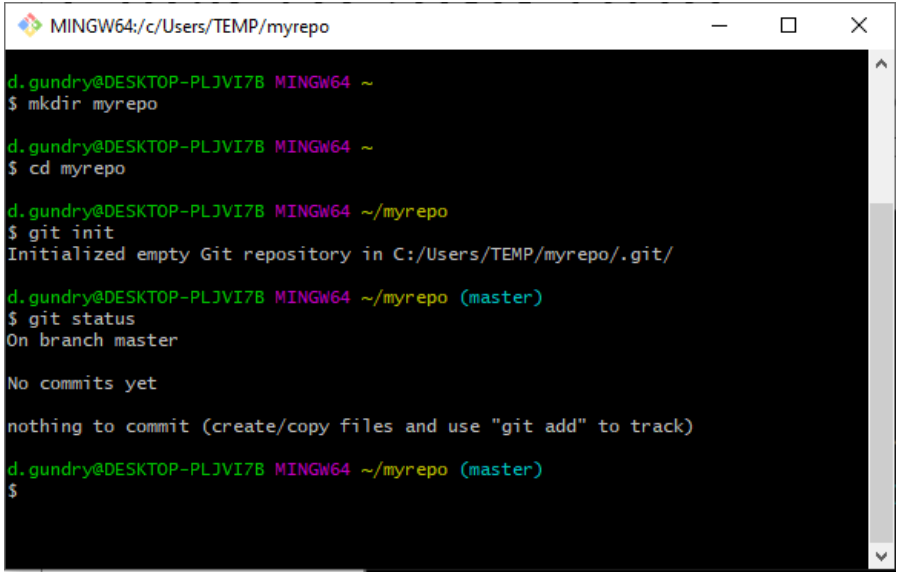
Create a local repository

- Here we're going to see
  - How to create a new repository from scratch
  - How to make our first commit

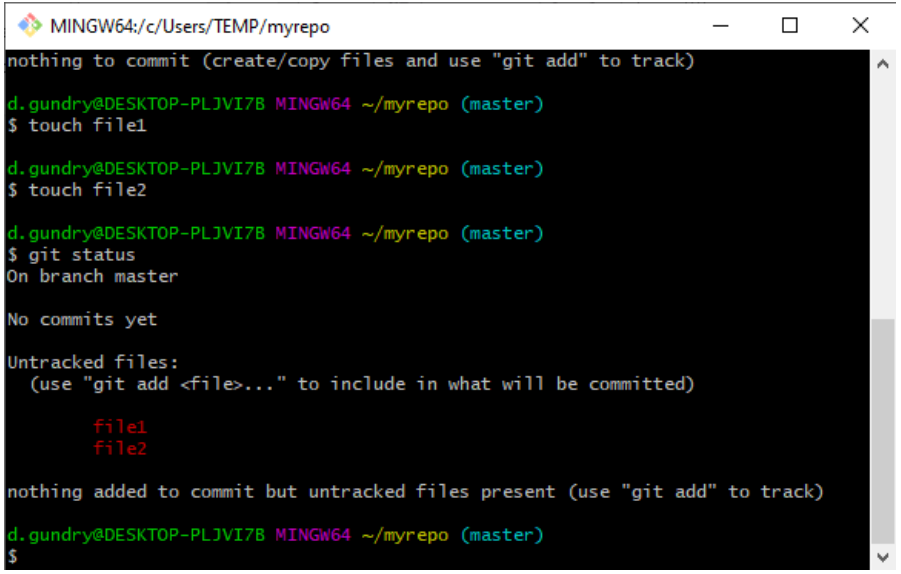


- Open your terminal emulator of choice
  - On Windows you can use Git Bash
- Navigate to wherever you want to store your project
  - Hint: use **`ls`** and **`cd`** commands to orient yourself and change directory

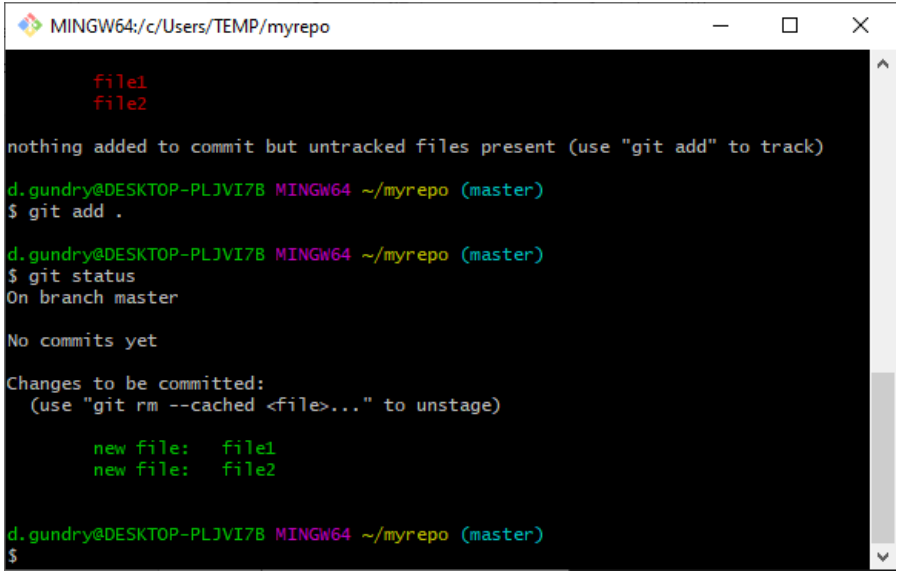
Create a repository



- Make a directory to store your repository
  - **`mkdir`**
- Enter that directory
  - **`cd`**
- Initialise a git repository
  - **`git init`**
- Check the status of the repository
  - **`git status`**



- Add some files. You can copy in your project files.
  - (I've used **`touch`** to create empty files for the demo)
- Check the status of the repository
  - **`git status`**
  - There are unstaged changes (shown in red)

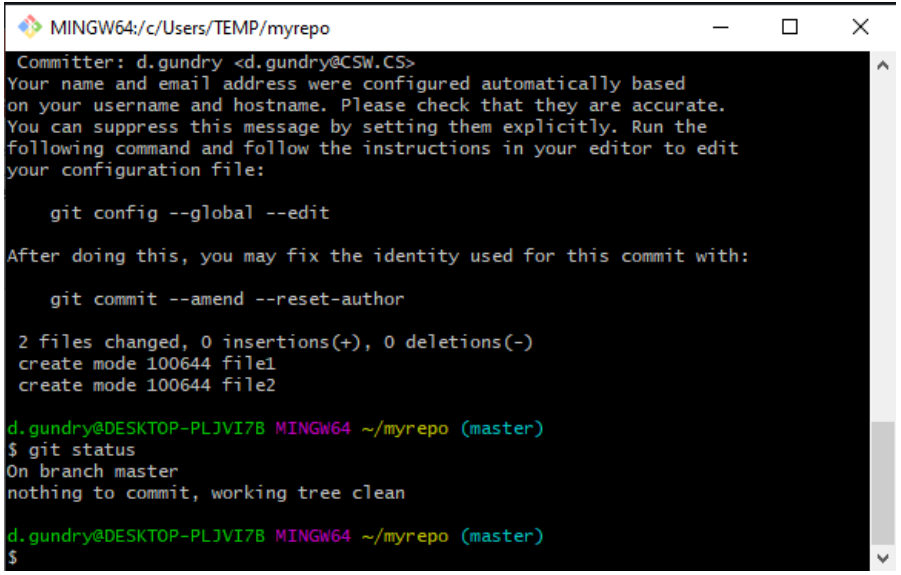


- Stage all changes
  - **`git add .`**
- Check the status of the repository
  - **`git status`**
  - There are staged changes (shown in green)

MINGW64:/c/Users/TEMP/myrepo

```
d.gundry@DESKTOP-PLJVI7B MINGW64 ~/myrepo (master)
$ git commit -m "Initial Commit"
[master (root-commit) c28ad89] Initial Commit
 Committer: d.gundry <d.gundry@CSW.CS>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1
 create mode 100644 file2

d.gundry@DESKTOP-PLJVI7B MINGW64 ~/myrepo (master)
$ |
```

- Commit the staged changes with a message
  - **`git commit -m “<message>”`**
  - It’s common to use “Initial commit” for the first commit message.



- Check the status of the repository
  - **`git status`**
  - There are no changes
  - All our changes have been committed

- **Task:**
  - Create a new repository and commit a file called README.md
  - Add another file called LICENSE.md and then make a second commit

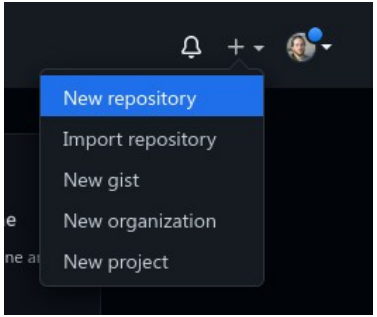
- git init
  - turn directory into a new repository
- git add .
  - stage all files for commit
- git status
  - show what files are staged for commit
- commit -m “Initial commit”
  - commit staged files

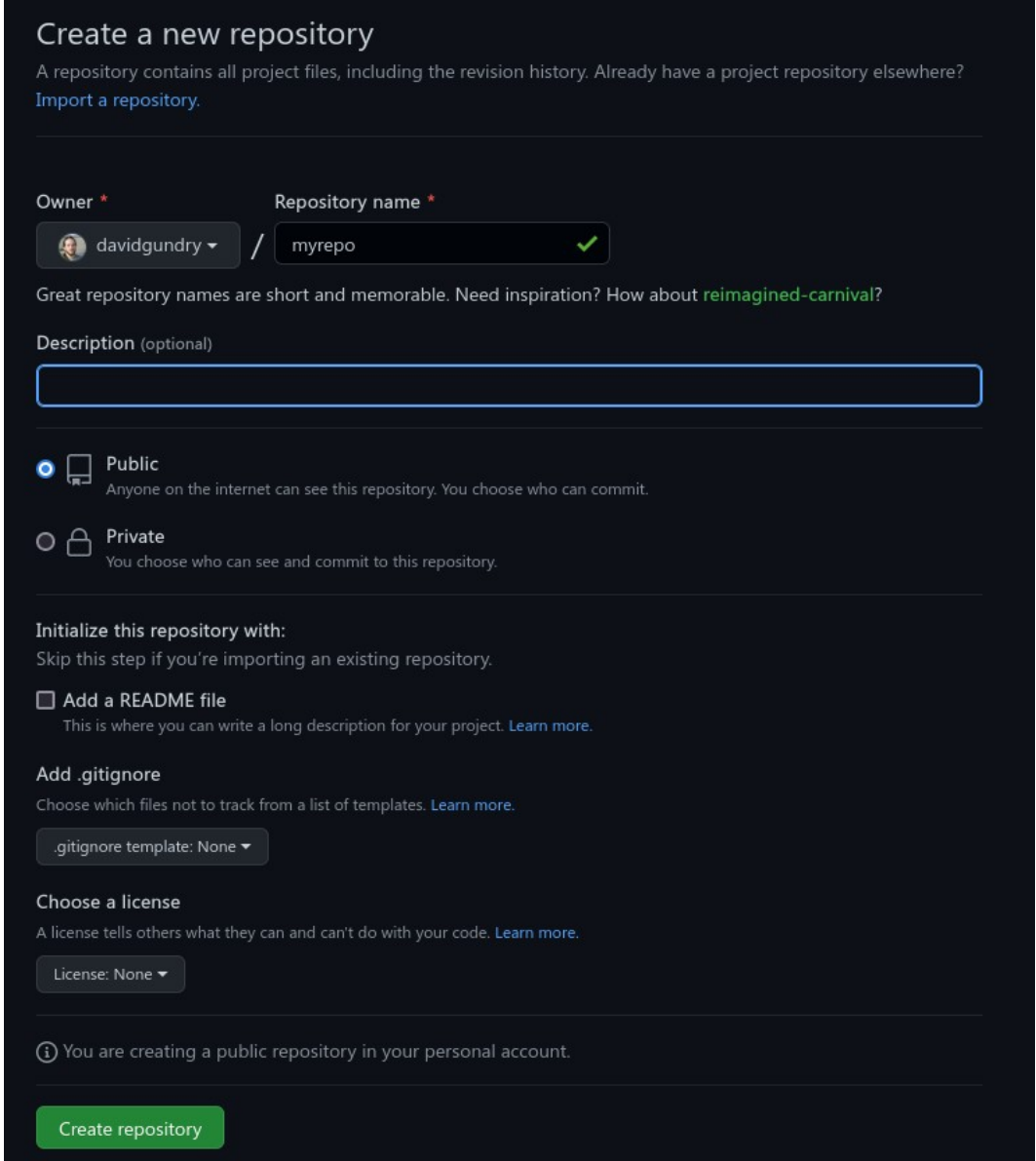
Set a remote and push commits

- Here we're going to see
  - How to associate our local repository with an empty remote repository
  - How to push our commits to the remote

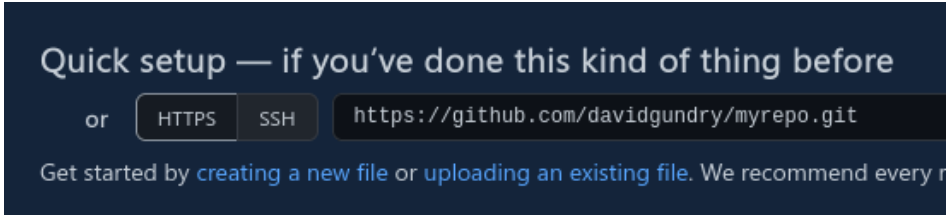
Set remote and push commits

- Go to GitHub and create an empty repository
  - Do not add readme, .gitignore or license as these will mean the repository is not empty

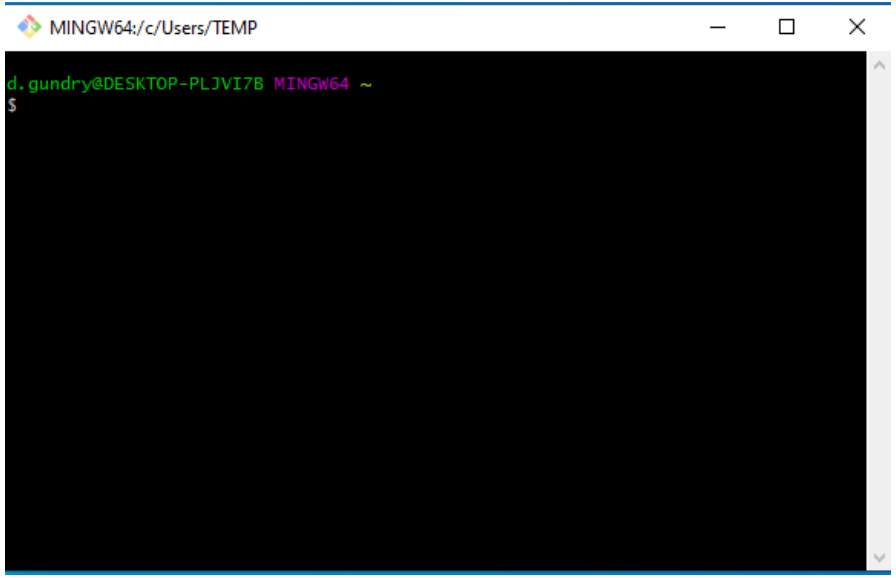




- Note the URL of your repository
  - ends in .git
- There are two options:
  - HTTPS (connect via HTTPS)
  - SSH (connect via SSH)
- The main difference is how your authenticate
  - HTTPS: personal access token
  - SSH: ssh key

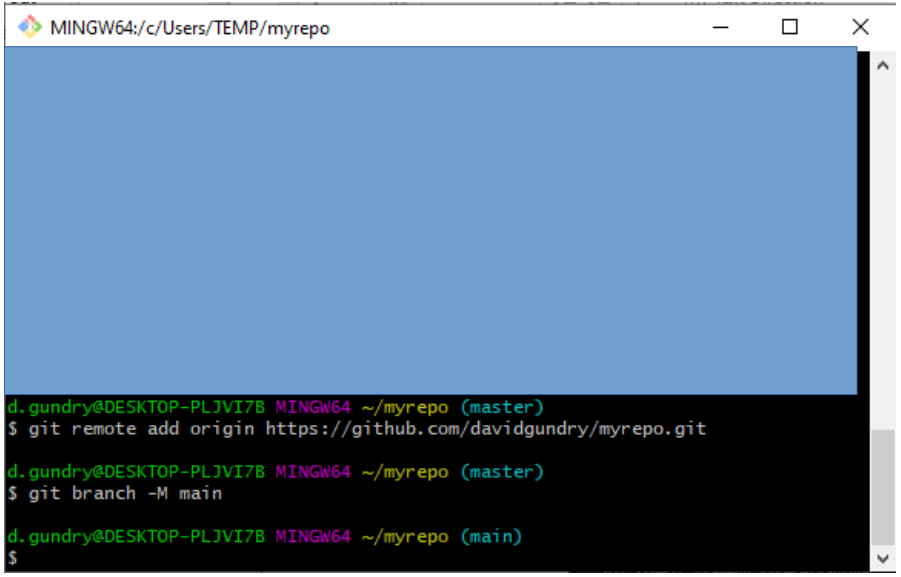


Set remote and push commits



- Open your terminal emulator of choice
- Navigate into your repository
  - Hint: use **`ls`** and **`cd`** commands to orient yourself and change directory

Set remote and push commits



- Add the GitHub repository as a remote called ‘origin’.
  - **`git remote add`**
- GitHub calls its main branch ‘main’ instead of ‘master’ Rename our current branch (master) to follow this standard
  - **`git branch -M main`**

Set remote and push commits

MINGW64:/c/Users/TEMP/myrepo

```
$ git status
On branch master
nothing to commit, working tree clean

d.gundry@DESKTOP-PLJVI7B MINGW64 ~/myrepo (master)
$ git remote add origin https://github.com/davidgundry/myrepo.git

d.gundry@DESKTOP-PLJVI7B MINGW64 ~/myrepo (master)
$ git branch -M main

d.gundry@DESKTOP-PLJVI7B MINGW64 ~/myrepo (main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 213 bytes | 213.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/davidgundry/myrepo.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.

d.gundry@DESKTOP-PLJVI7B MINGW64 ~/myrepo (main)
$
```

- Push our commits
  - **`git push`**
- The first time we do this, we need to tell our main branch where it needs to push to
  - **`git push -u origin main`**
  - Our ‘upstream’ is set to the main branch on our origin remote

- Summary
  - In repository, run:

    git remote add origin <repository-url> (add a remote)

    git branch -M main (name branch to GitHub's default)

    git push -u origin main (push, setting upstream for branch)

Other useful commands

- git branch <name>
  - Create a new branch called <name>
- git checkout <branch>
  - Switch to the branch called <branch>
- git merge <branch>
  - Merge the named branch into the current branch
- git reset
  - By itself unstages all staged files
- git log
  - Show a history of commits
- git commit –amend
  - Add currently staged changes to the previous commit