

- IMAGE CONSOLE SANDBOX -

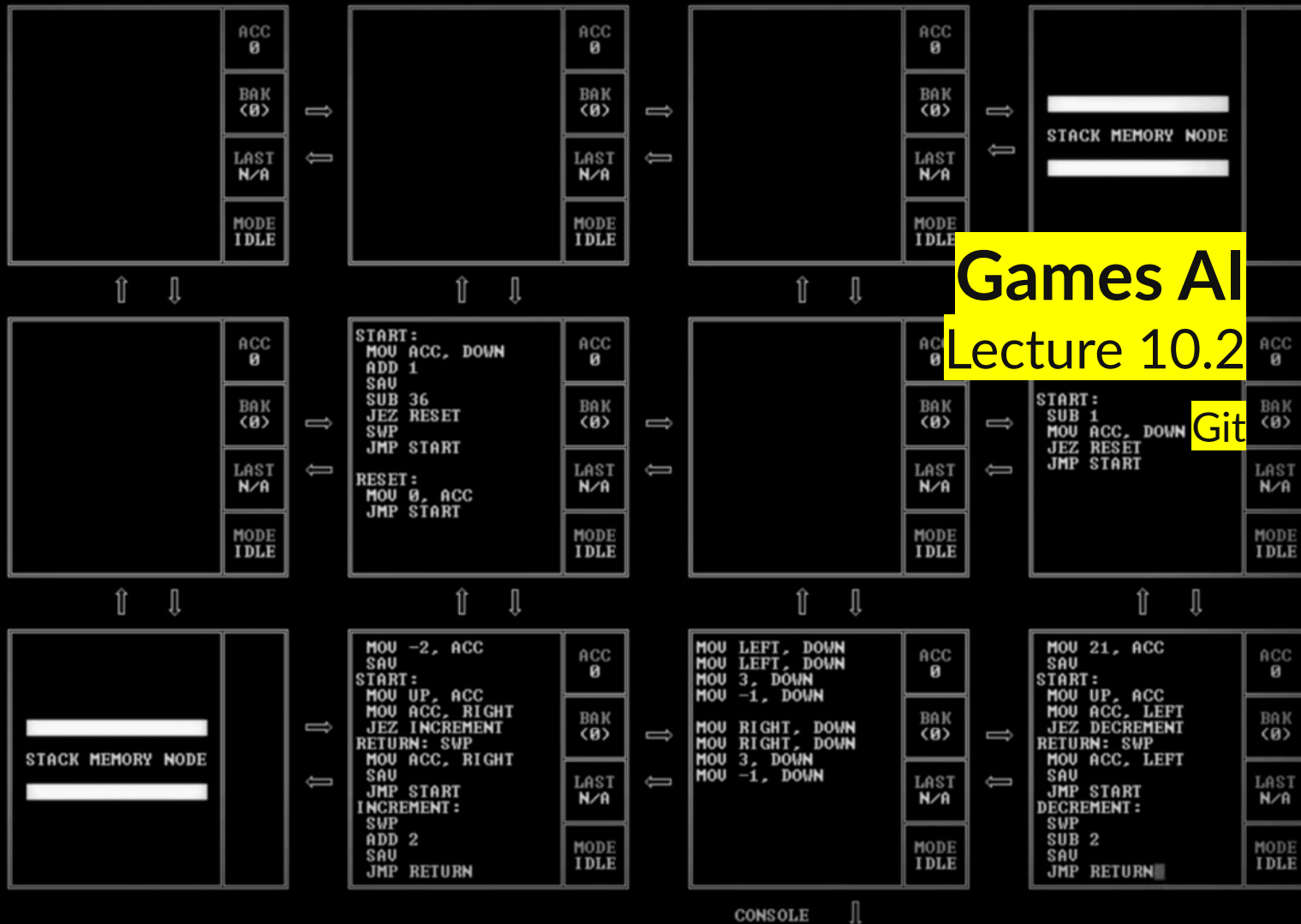
CONSOLE

CONSOLE

# Games AI

## Lecture 10.2

Git



7

8

9

4

5

6

1

2

3

0

ENTER

STOP

STEP

PLAY

FAST

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

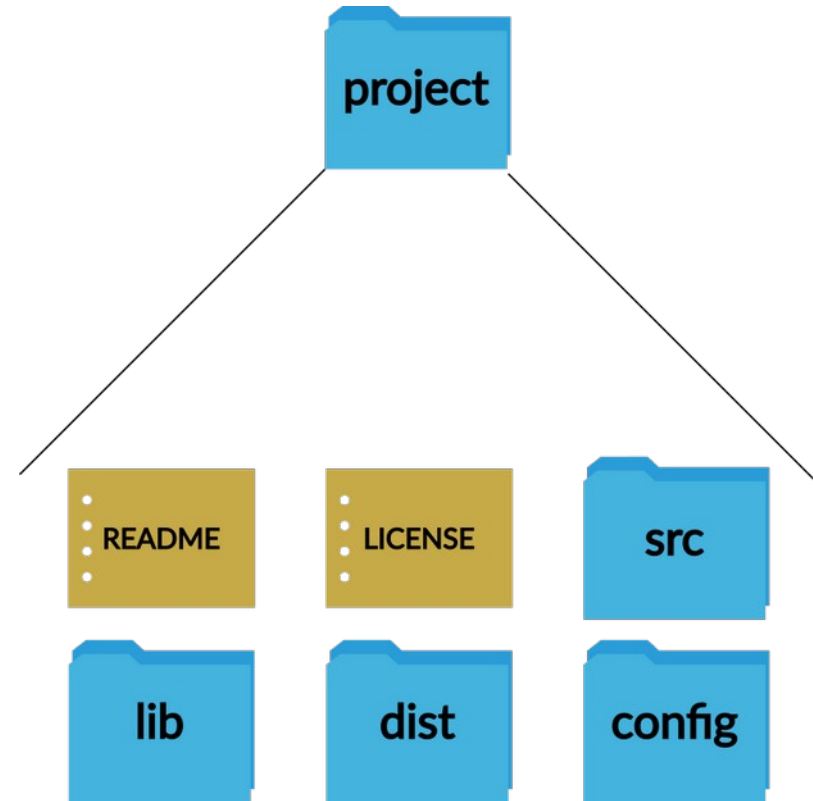
Why Git



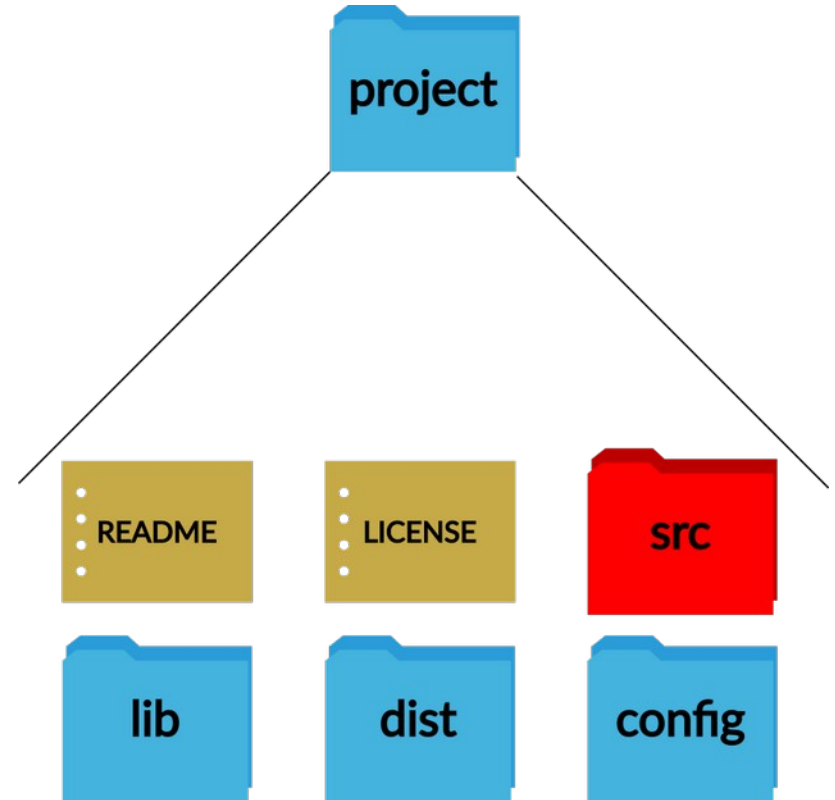
- You have a project folder on your computer



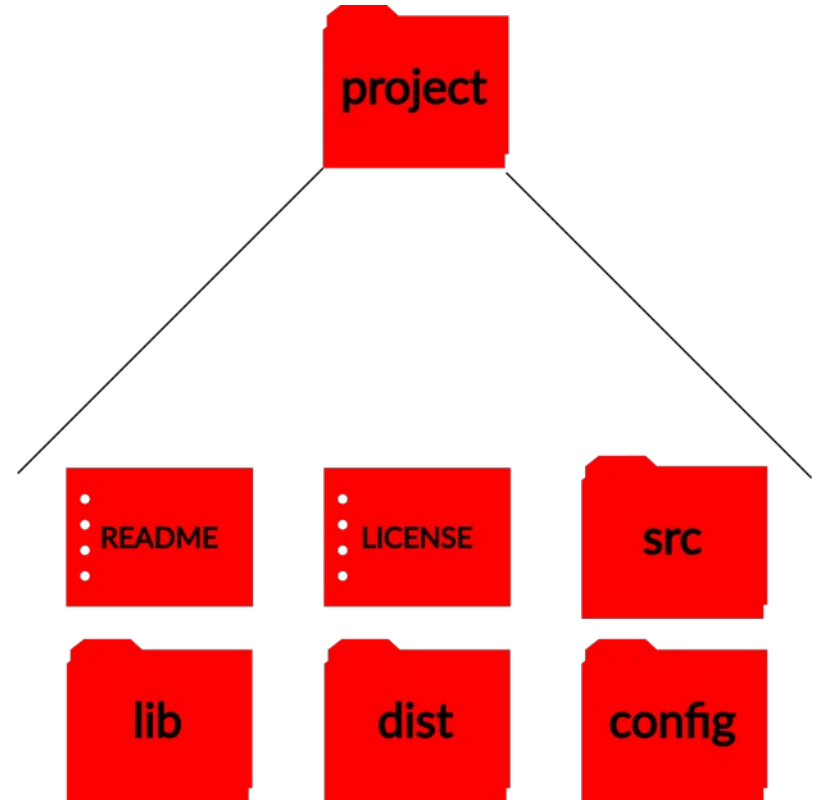
- Inside are the files for your project
  - Source code
  - Libraries
  - Built code
  - Configuration files
  - Assets



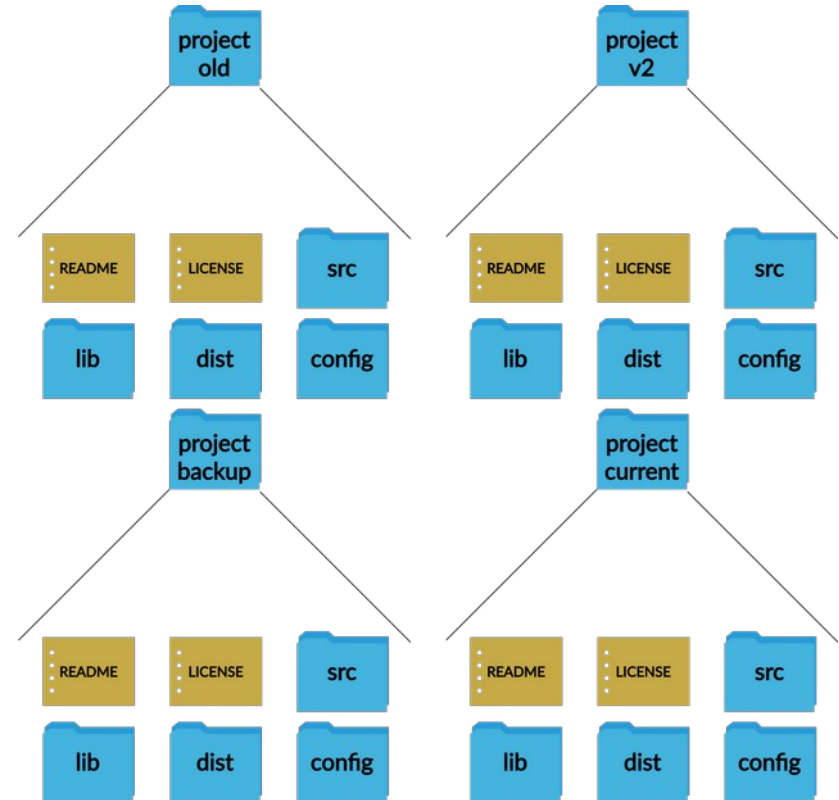
- What happens if
  - You delete a file?



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

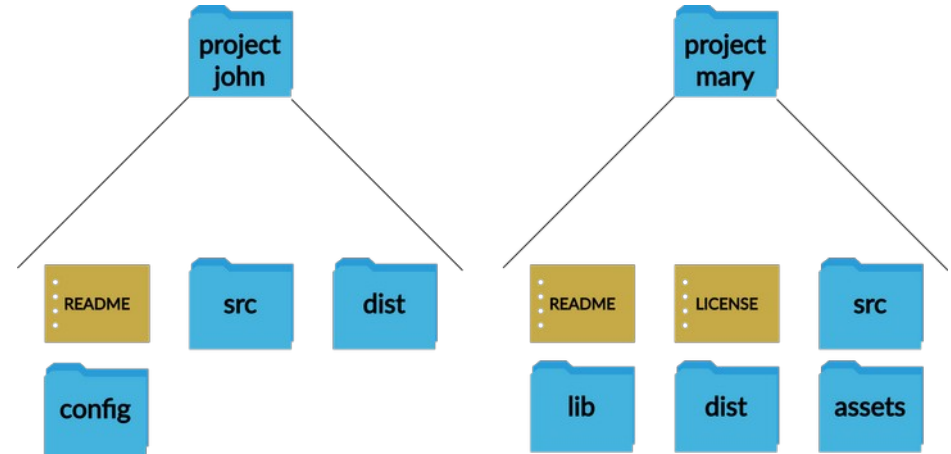


- What happens if
  - You delete a file?
  - You delete the whole folder?
  - You backup the folder?

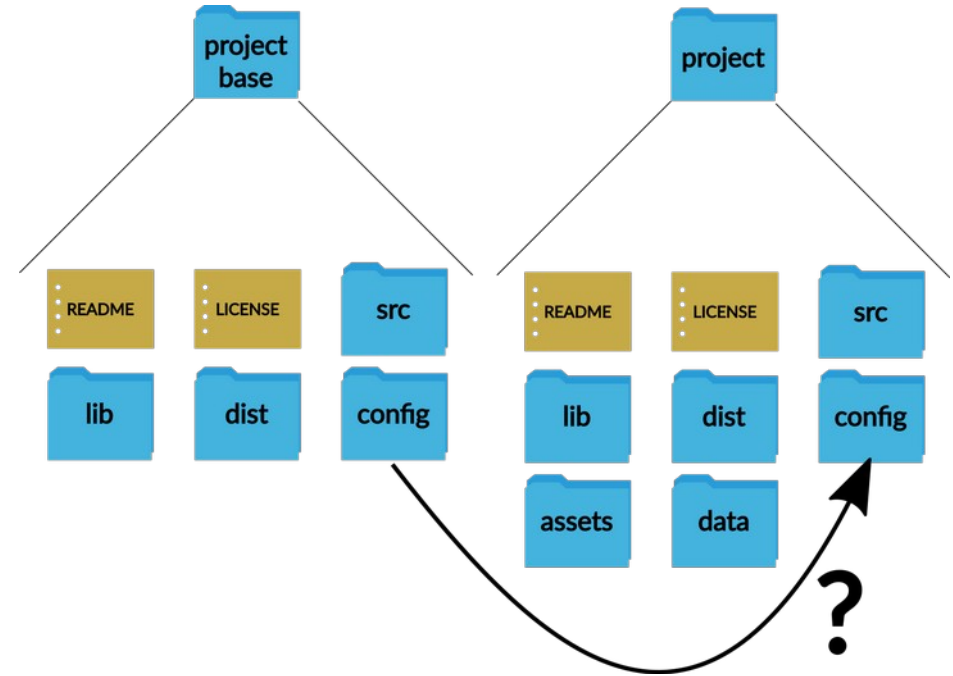




- What happens if
  - You delete a file?
  - You delete the whole folder?
  - You backup the folder?
  - A team works on the same project?



- 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



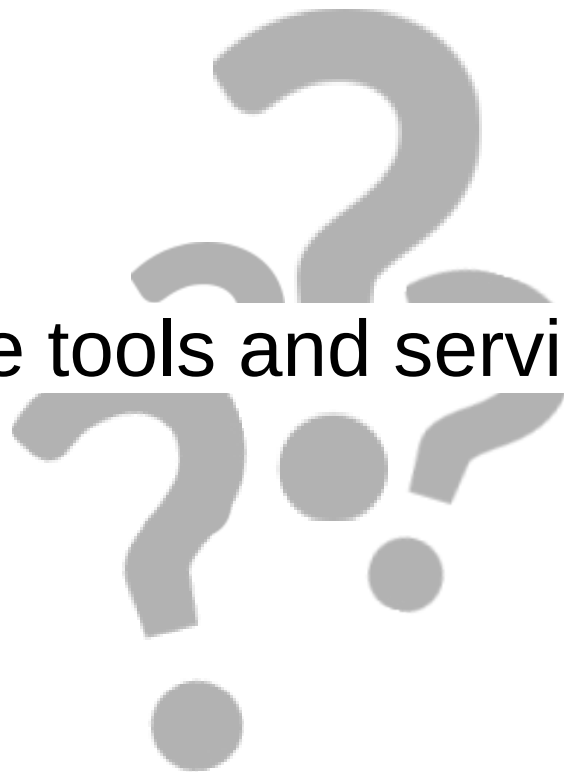
- What happens if
  - You want to backup a team project that adapts an existing project, and then incorporate updates to the base project while simultaneously developing new features
  - **AND** have a central server that regularly runs tests on everyone's contributions
  - **AND** automatically build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

**And do all this with freely available tools and services**  
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
  - **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

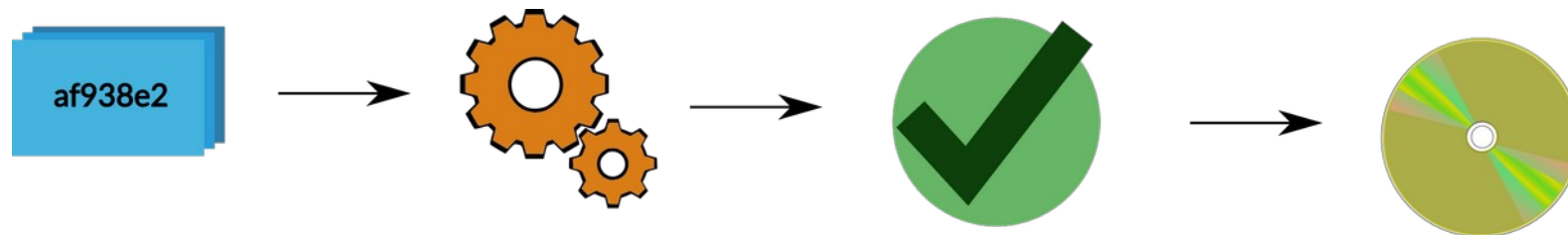


- Workflow is important
  - Your time is important
  - Your enthusiasm is important
- Bad workflow leads to bad projects
  - Poor code quality
  - Cutting corners
  - Slow updates
  - Integration hell

- Good workflows make development easier

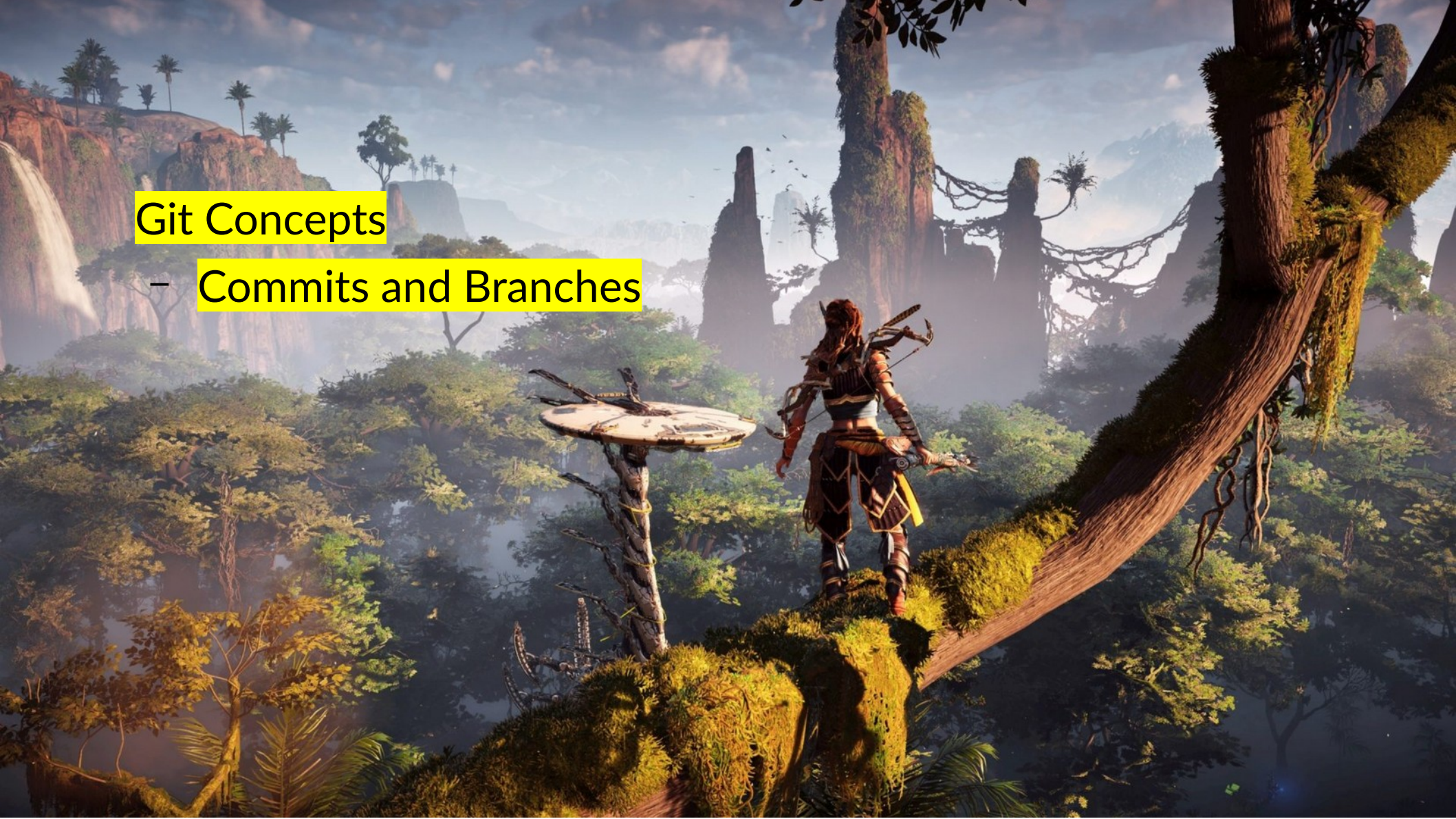


- Continuous Delivery

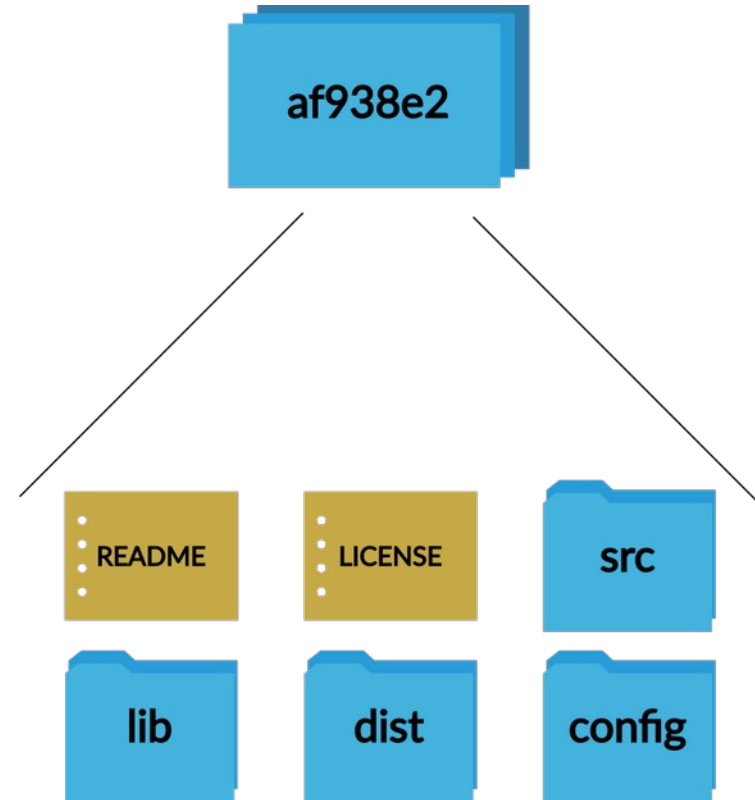


# Git Concepts

- Commits and Branches

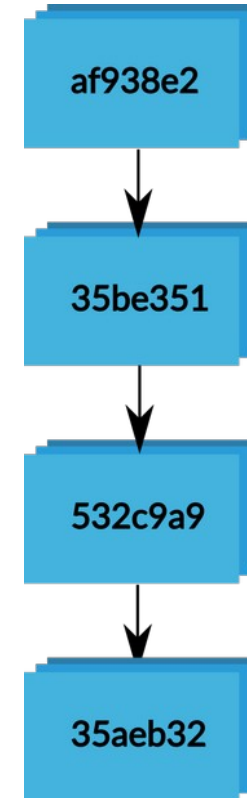


- Commit (noun)
  - A snapshot of your project





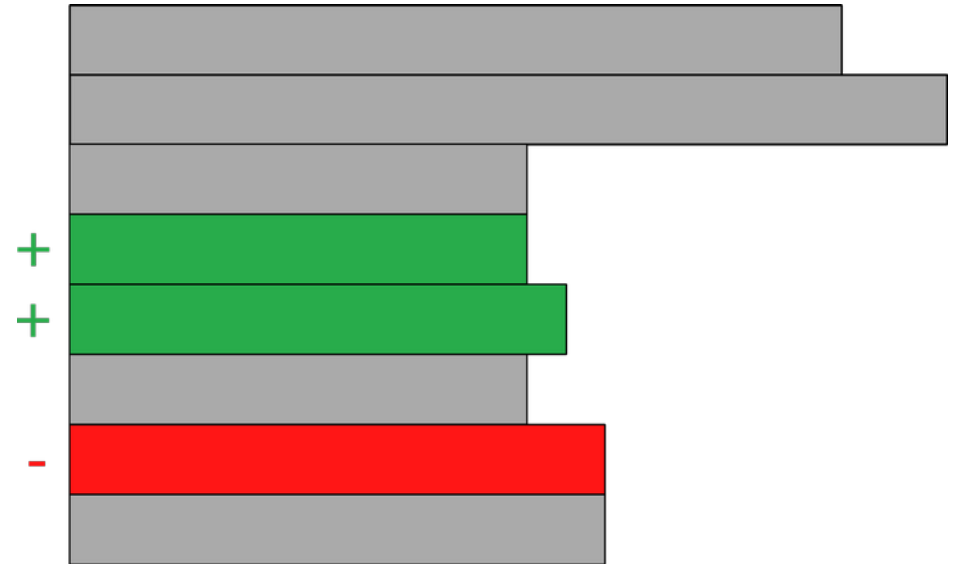
- Branch (noun)
  - A sequence of commits
  - Shows how your project has changed over time



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

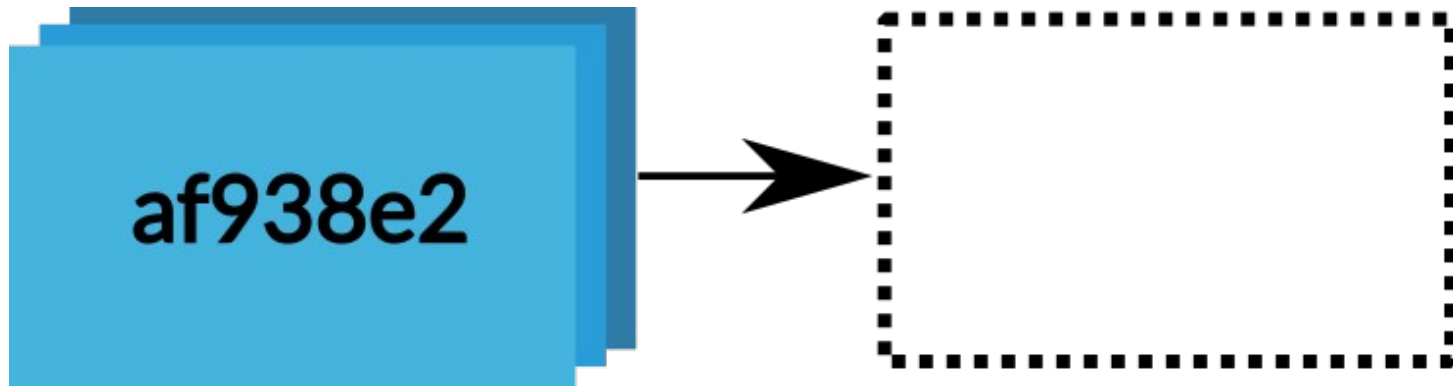


- Commits
  - Commits are calculated line-by-line
  - A commit stores just the changes that have been made since the last commit

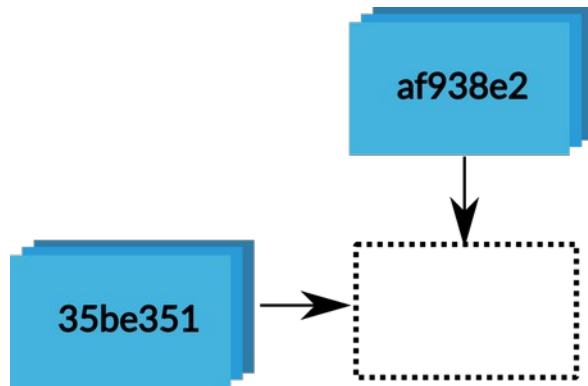




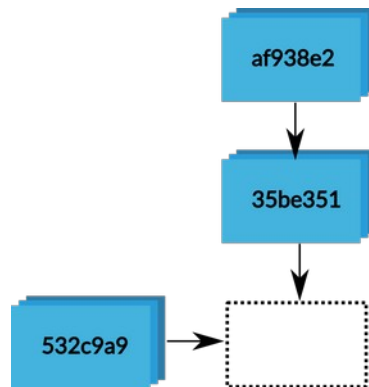
- Simple workflow example
  - Create a new repository and make an initial commit



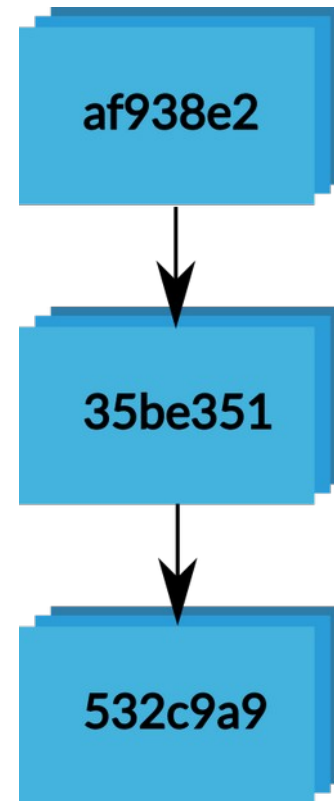
- Simple workflow example
  - Create a new repository and make an initial commit
  - Do some work, then commit it



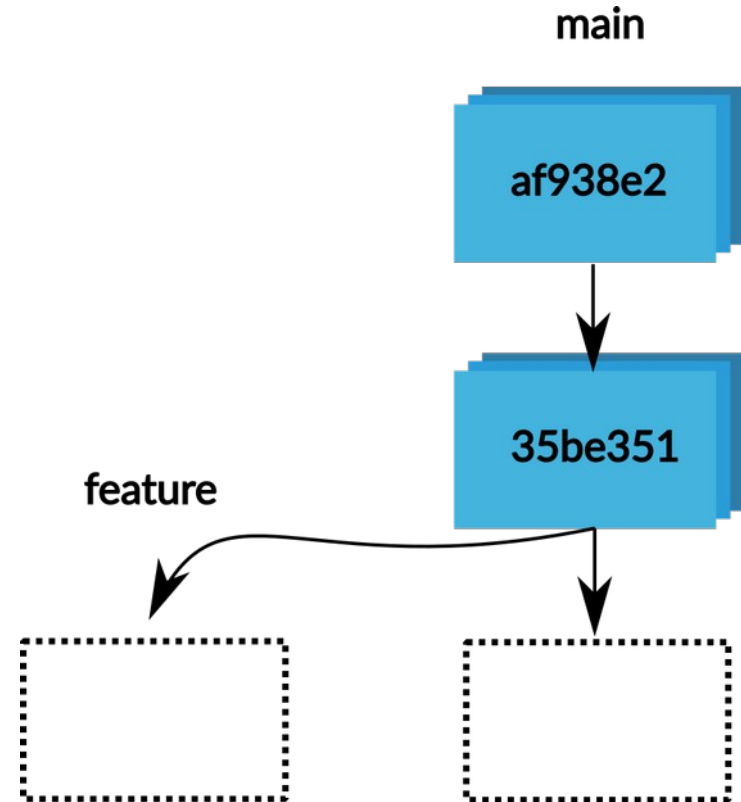
- Simple workflow example
  - Create a new repository and make an initial commit
  - Do some work, then commit it
  - Do some more work, then commit it



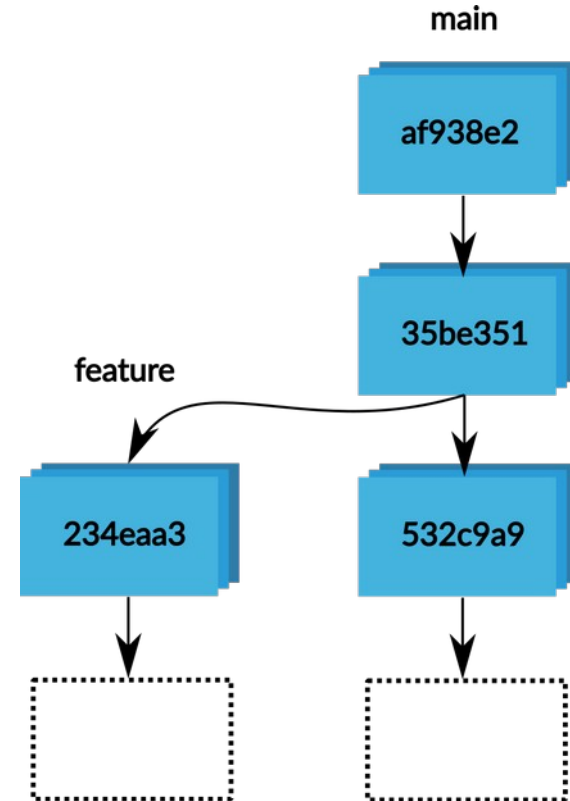
- Simple workflow example
  - Now you've got a branch with
    - The current state of your project
    - All changes made to your project



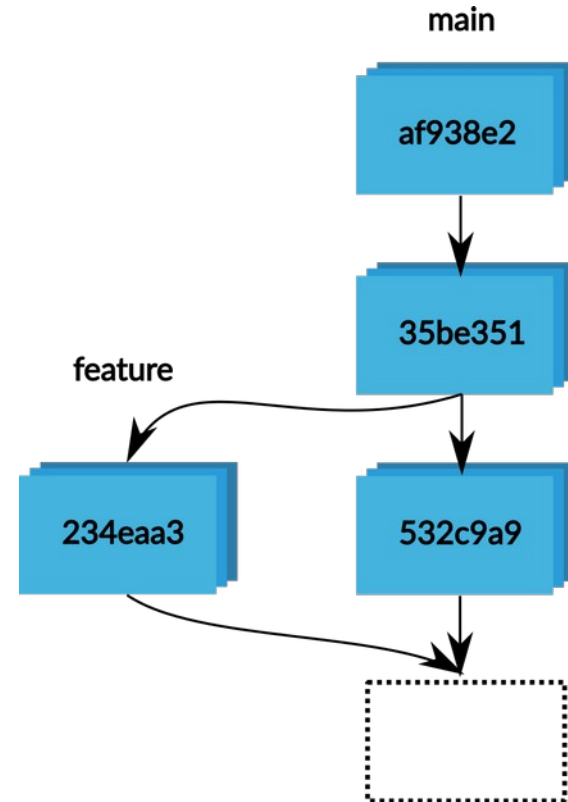
- Branch (verb)
  - Take a branch and split a new one off it
  - Now you've got two branches



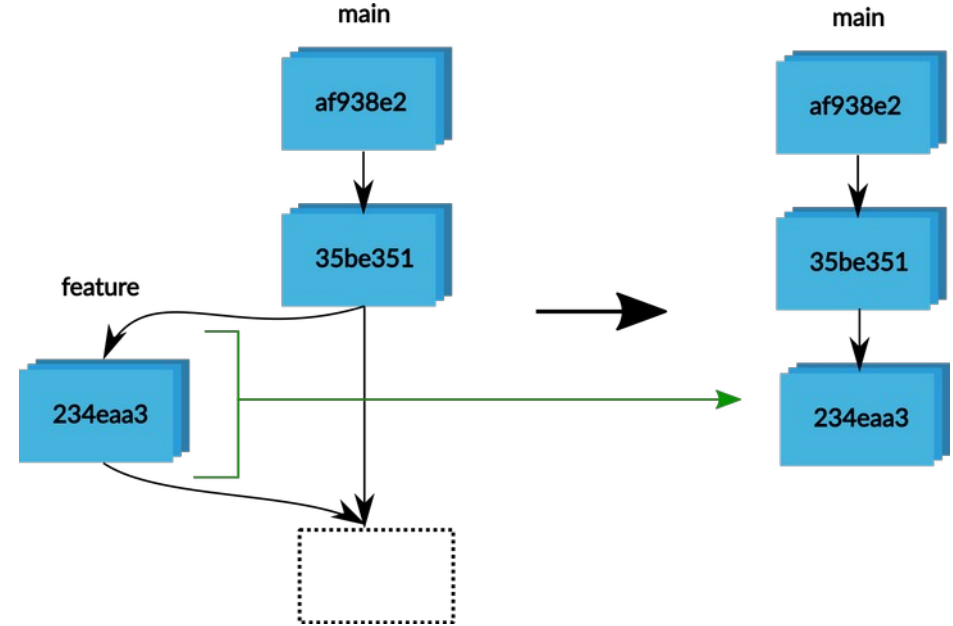
- Multiple branches
  - You can add commits to each branch
  - This gives you two different versions of your project



- Merge
  - Combine the changes made in two branches



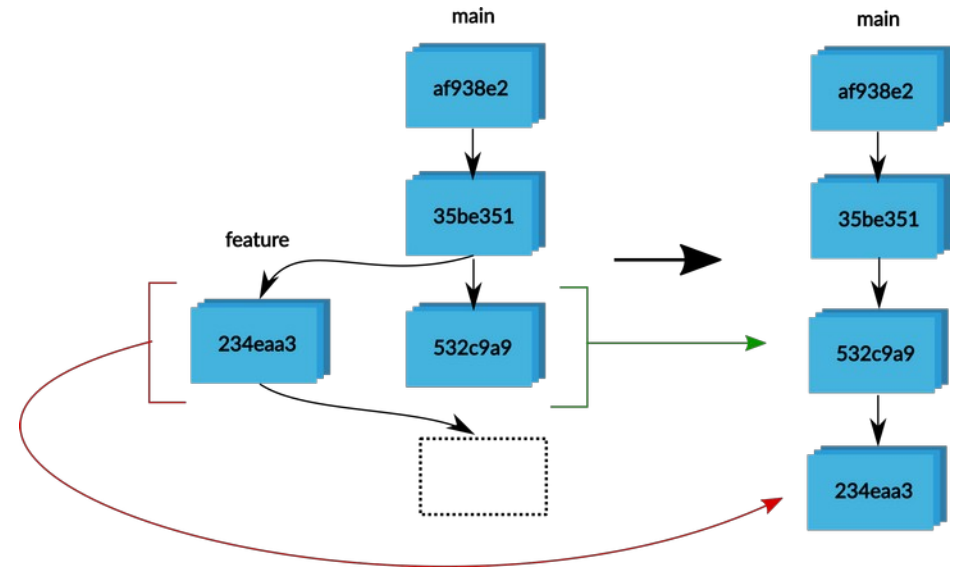
- Merge (Fast Forward)
  - There are only commits on the branch to merge in
  - These commits can be added on to the end of the main branch



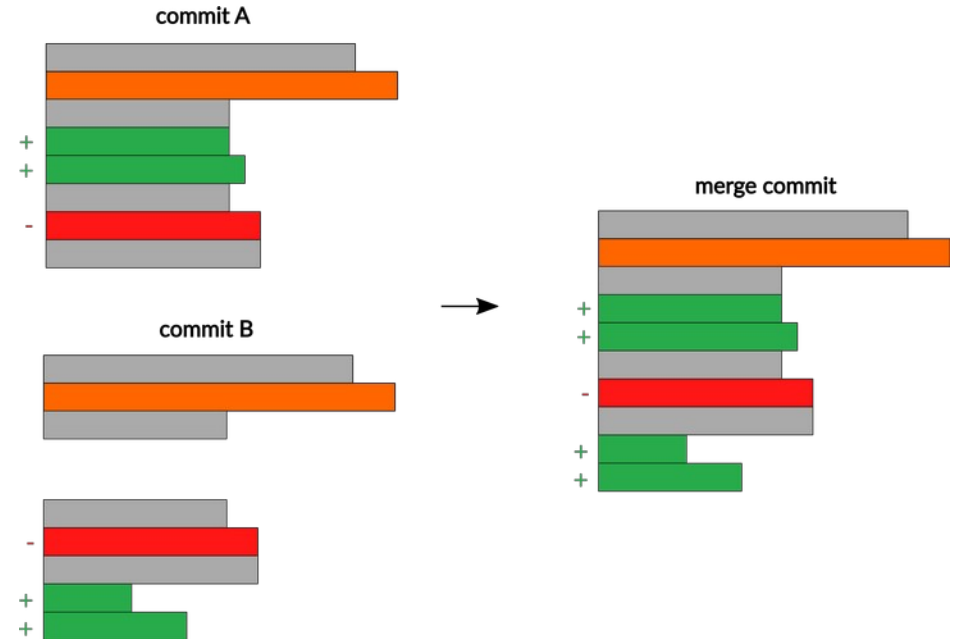




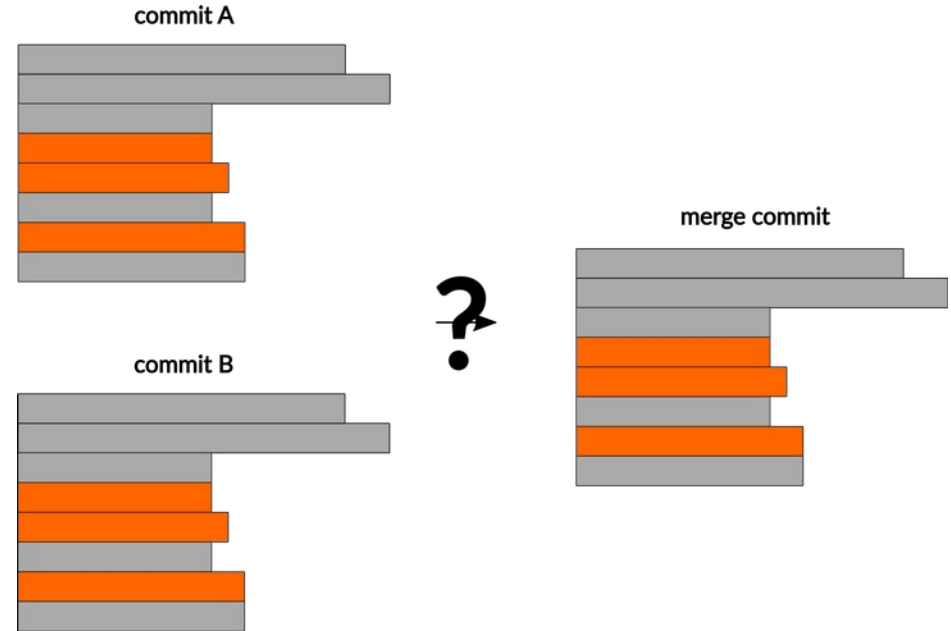
- Merge (Rebase)
  - Replay the feature branch on top of the new commits on the main branch



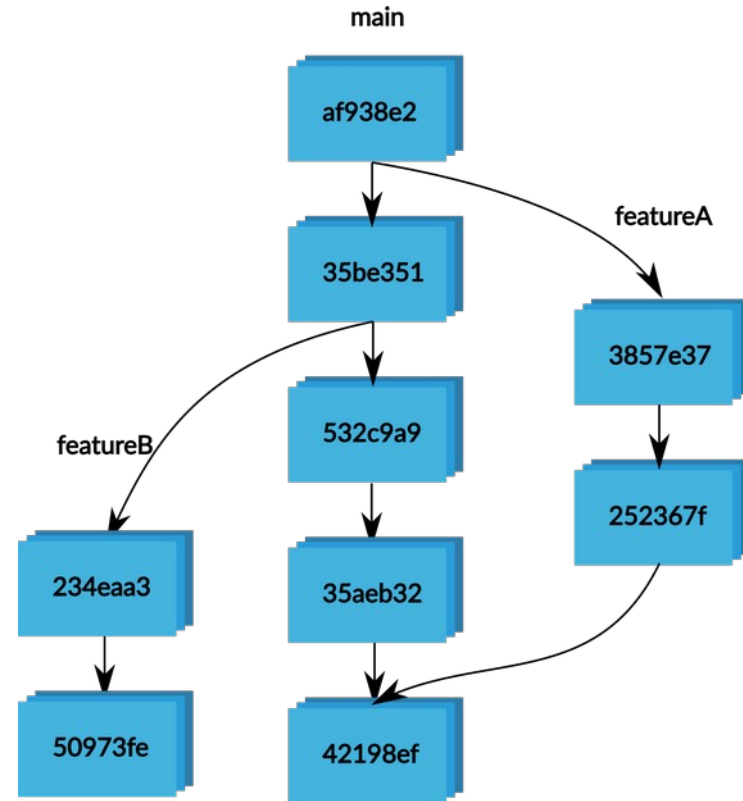
- Resolving a merge
  - Git merges line-by-line
  - If the branches change different lines everything resolves automatically



- 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

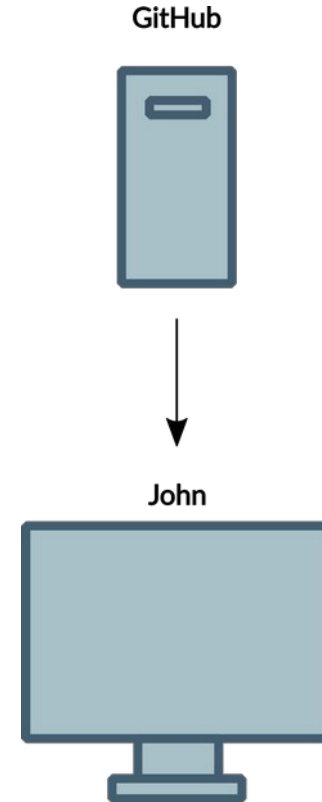


Git Concepts

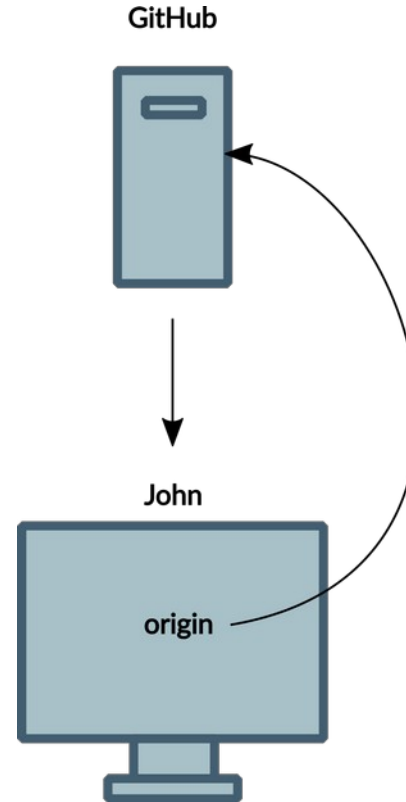
Working with remotes



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

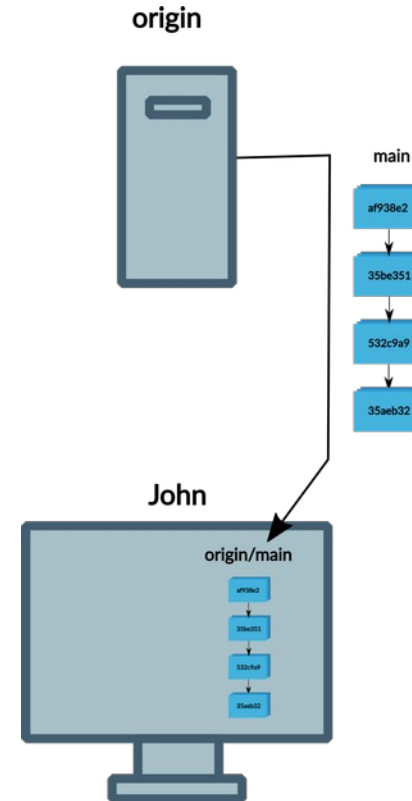


- Remote (noun)
  - A reference in your git repo to a repository elsewhere
  - When you clone a repo, it creates a remote called origin

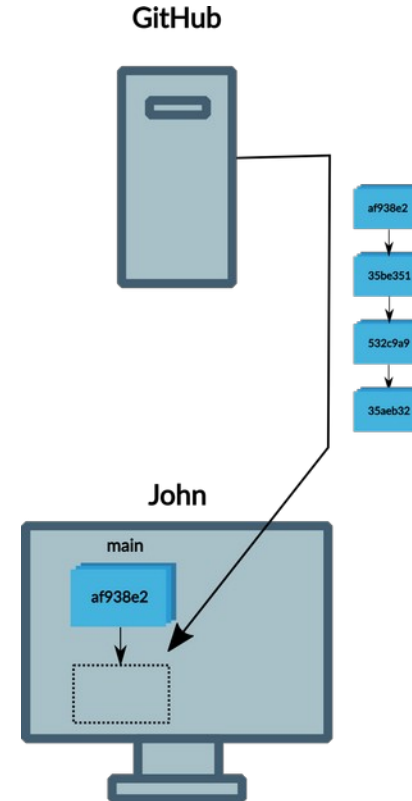




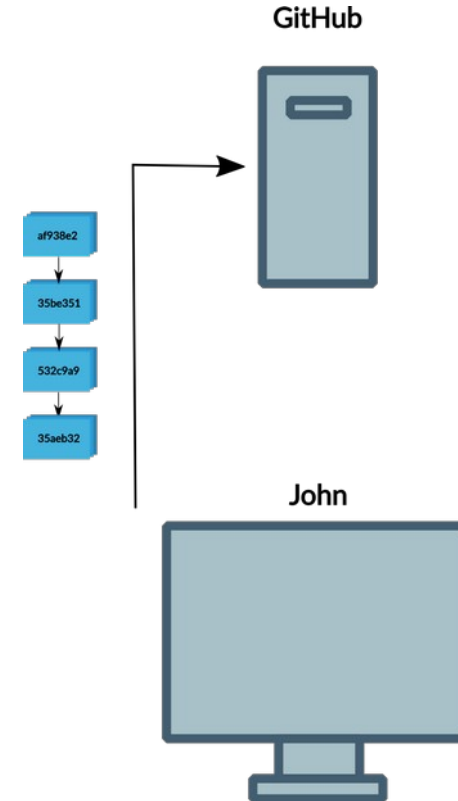
- Fetch (verb)
  - Download commits made on a remote branch
  - Fetching branch main from origin would put result in branch origin/main



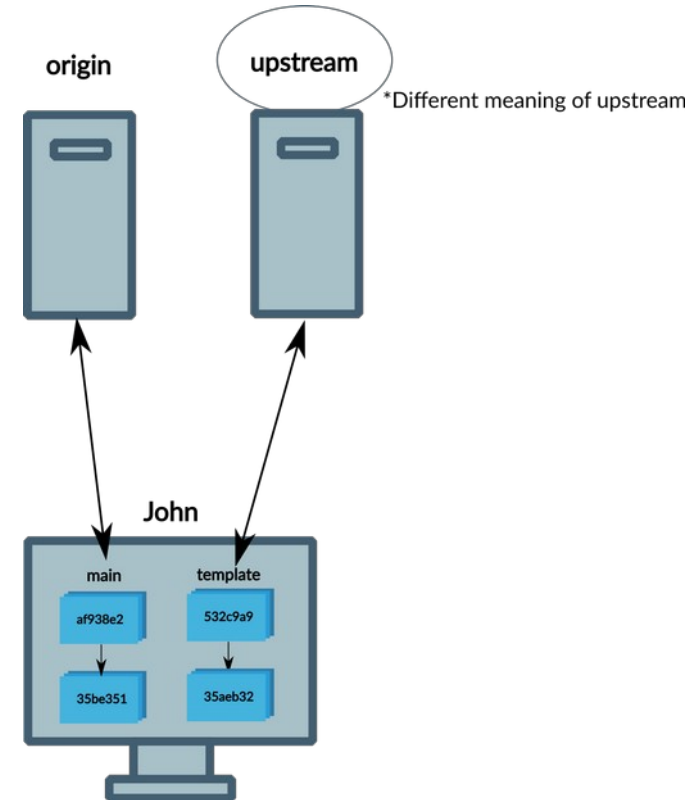
- Pull (verb)
  - Fetch **and merge** commits made on a remote branch



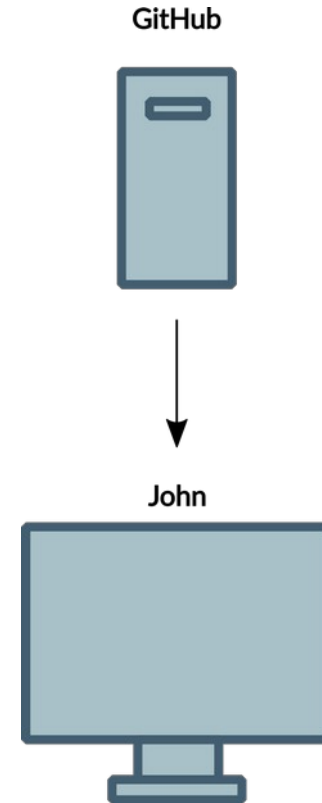
- Push (verb)
  - Upload new commits in the local repository to a remote branch



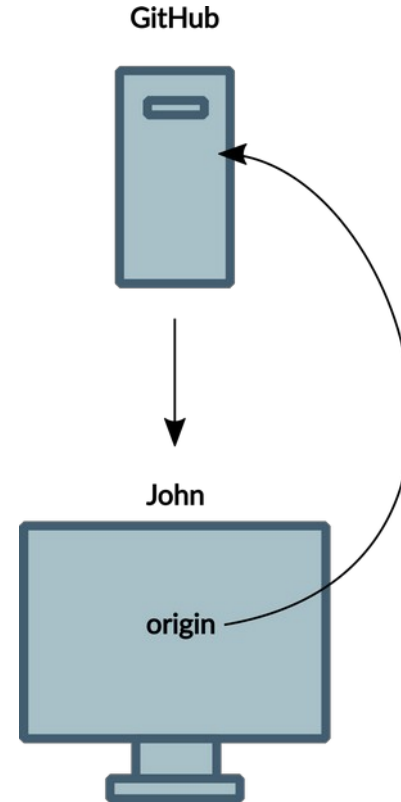
- Upstream (noun)
  - Which remote branch our branch should fetch/pull from and push to
  - Different branches may have upstreams on different remotes



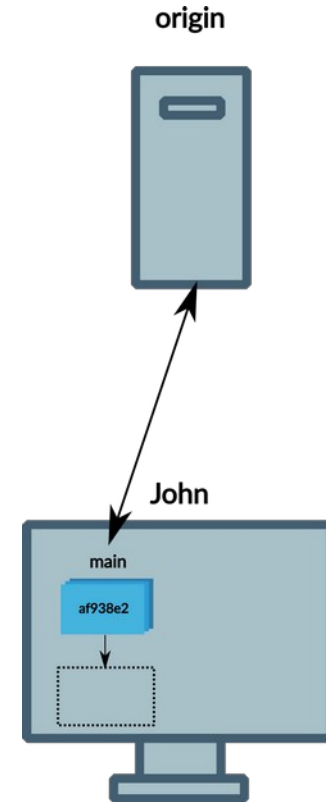
- Example workflow
  - Clone a repository



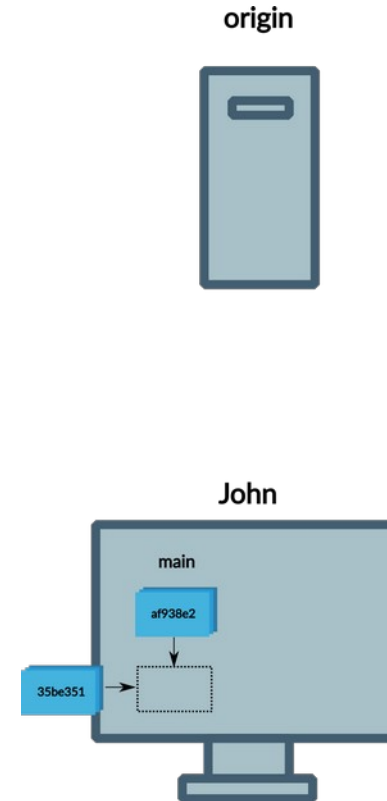
- Example workflow
  - Clone a repository
    - Remote **origin** is set automatically



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

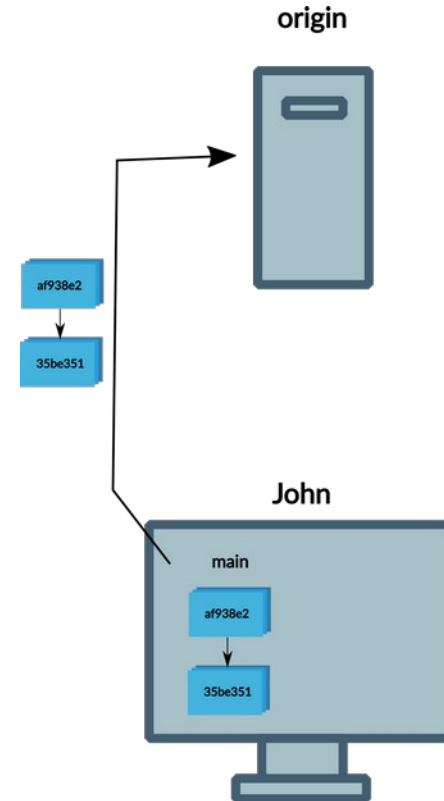


- Example workflow
  - Clone a repository
    - Remote **origin** is set automatically
  - Set upstream for branch **main**
  - Commit changes on branch **main**





- 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



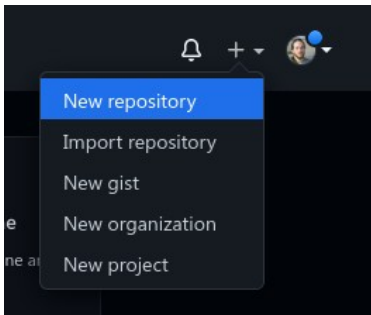


Clone a repository

- 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

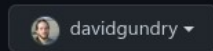


## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)

Owner \*



Repository name \*



Great repository names are short and memorable. Need inspiration? How about [reimagined-carnival?](#)

Description (optional)



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

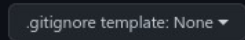


Add a README file

This is where you can write a long description for your project. [Learn more.](#)

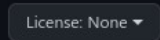
Add .gitignore


Choose which files not to track from a list of templates. [Learn more.](#)



Choose a license

A license tells others what they can and can't do with your code. [Learn more.](#)

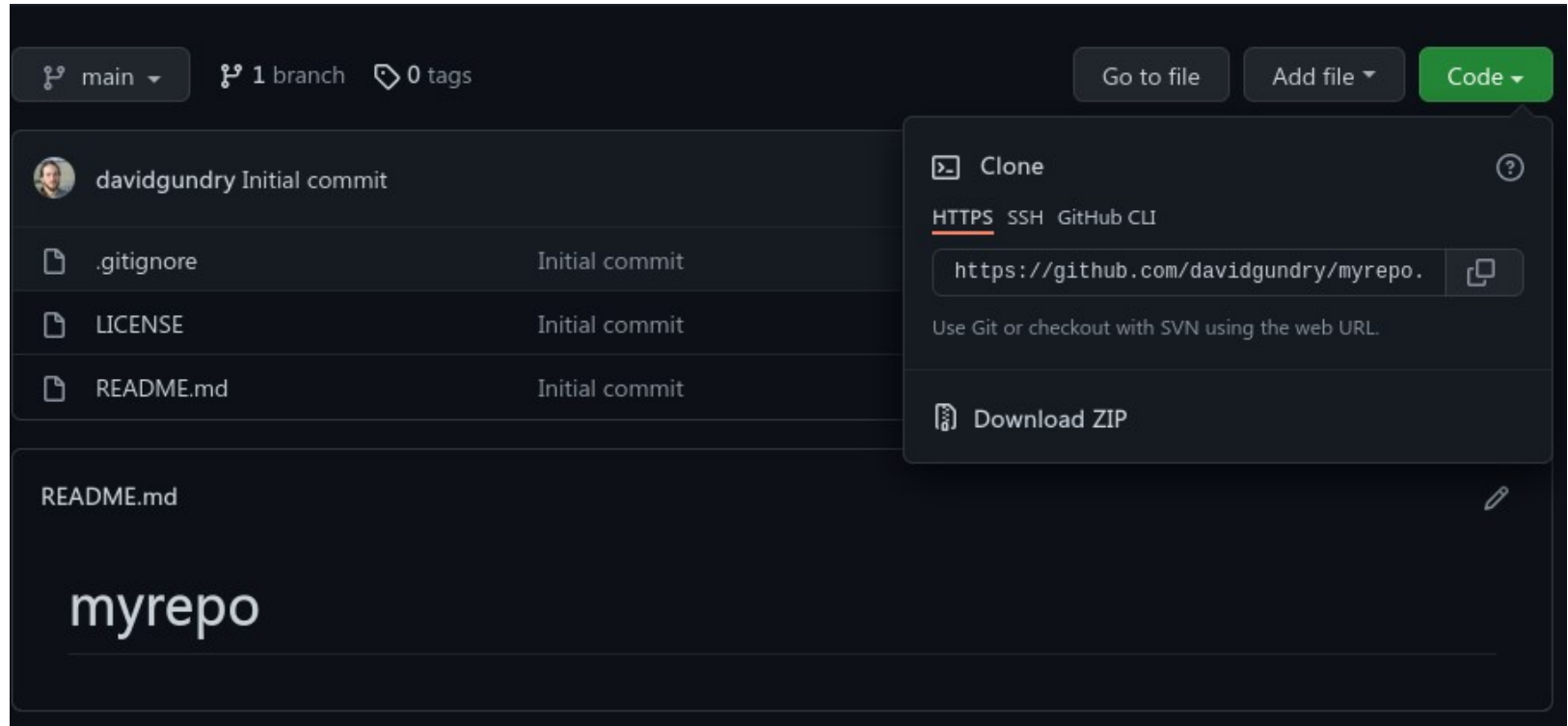


 You are creating a public repository in your personal account.

Create repository

## Clone a repository

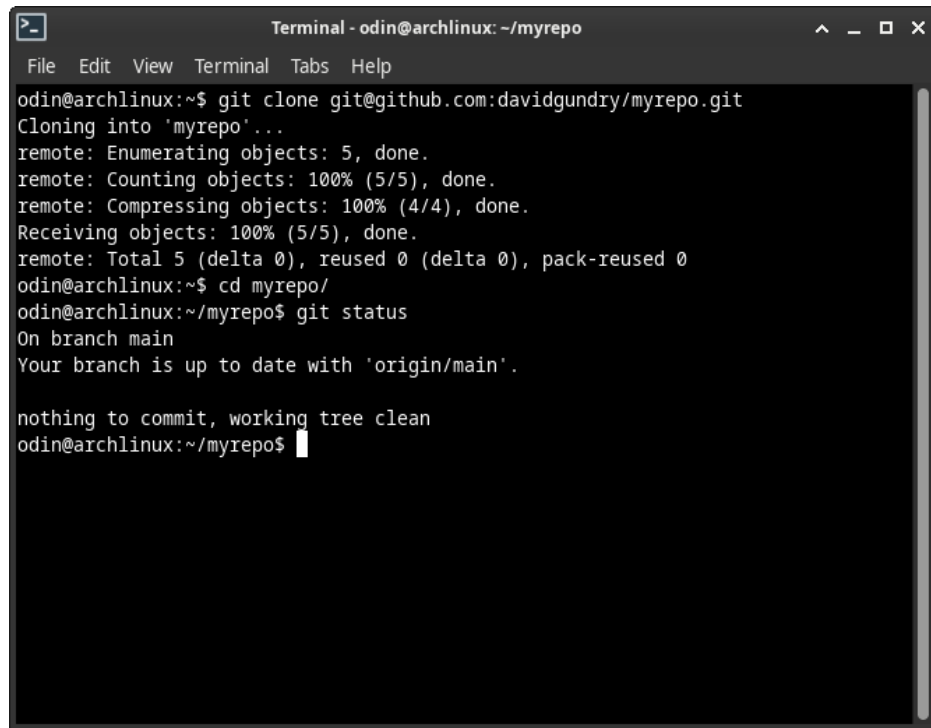
- Find the URL of the repository





## Clone a repository

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

A terminal window titled "Terminal - odin@archlinux: ~/myrepo" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows the following commands and output:

```
odin@archlinux:~$ git clone git@github.com:davidgundry/myrepo.git
Cloning into 'myrepo'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
Receiving objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
odin@archlinux:~$ cd myrepo/
odin@archlinux:~/myrepo$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
odin@archlinux:~/myrepo$
```

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

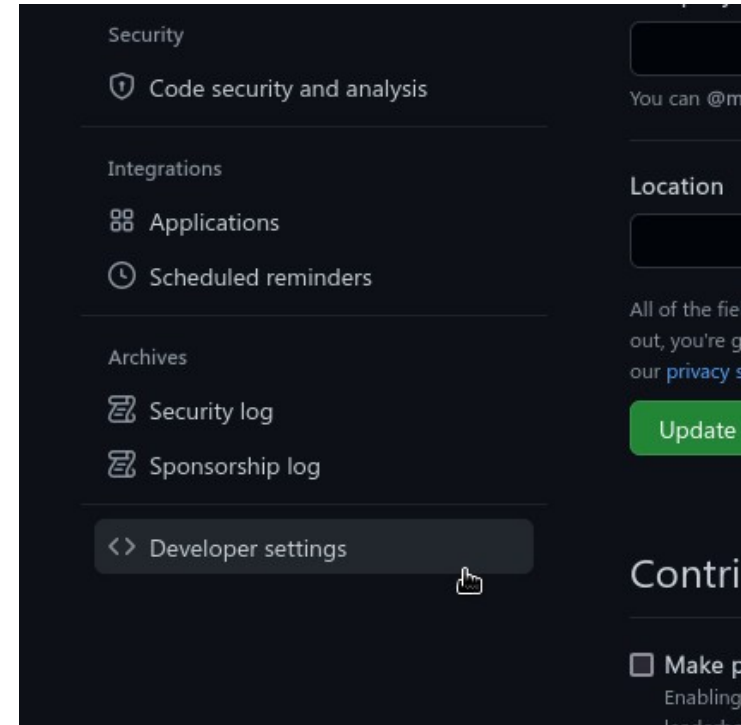
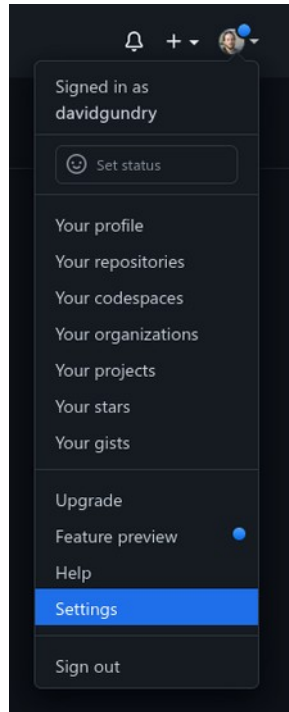
# HTTPS authentication with Personal Access Token



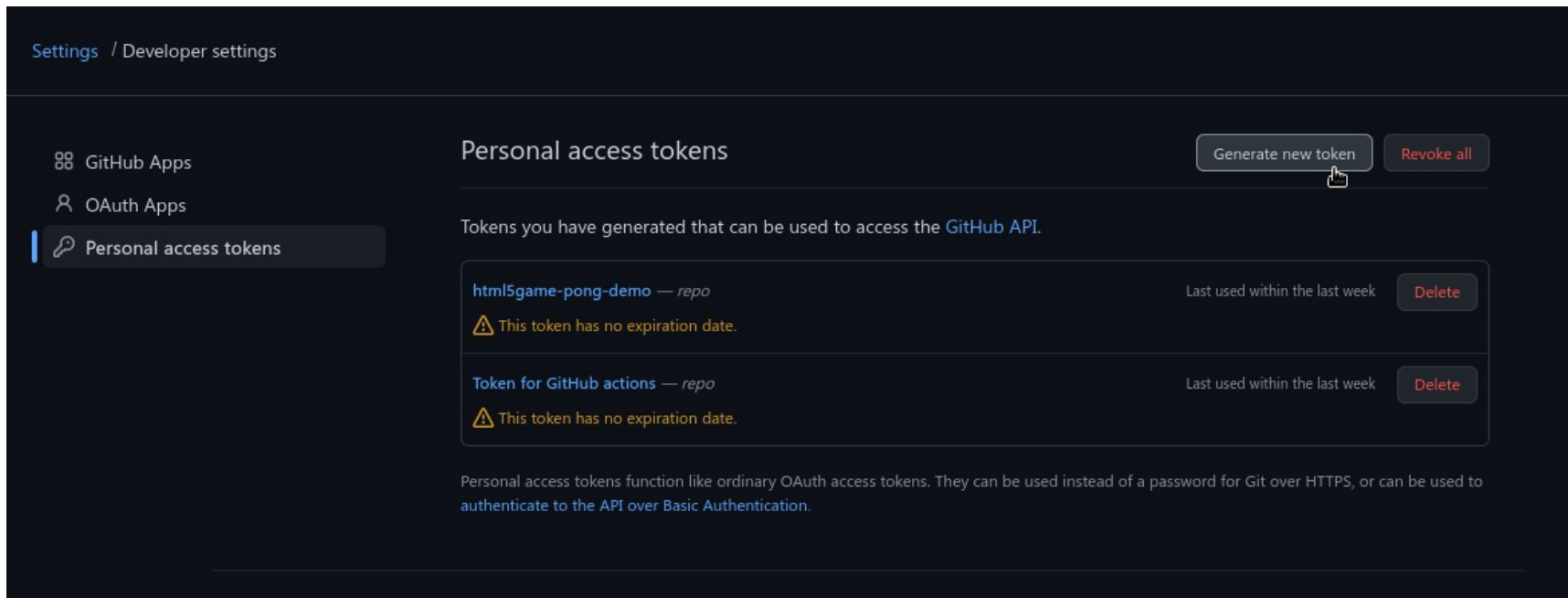
- GitHub doesn't allow authentication with username and password
- One method is to create a Personal Access Token
  - This is basically your password, so keep it safe



- Go to GitHub Settings > Developer Settings



- Click generate new token



- Give the token control over repositories

### New personal access token


Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

**Note**

For console authentication

What's this token for?

**Expiration \***

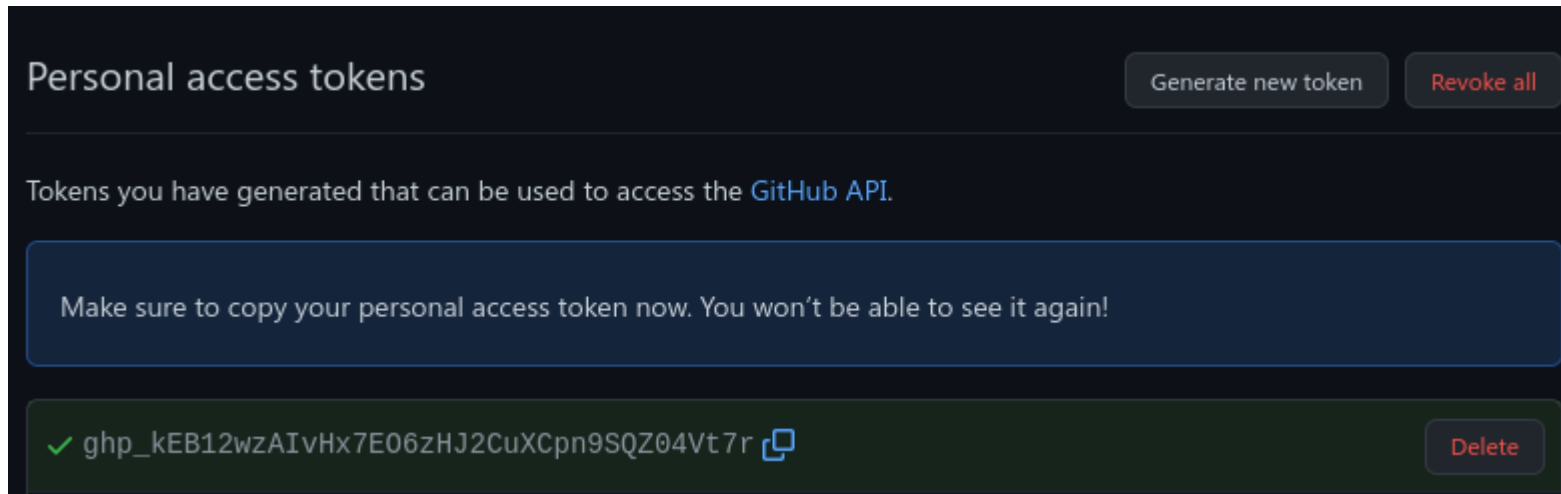
30 days  The token will expire on Sat, May 28 2022

**Select scopes**

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events

- 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

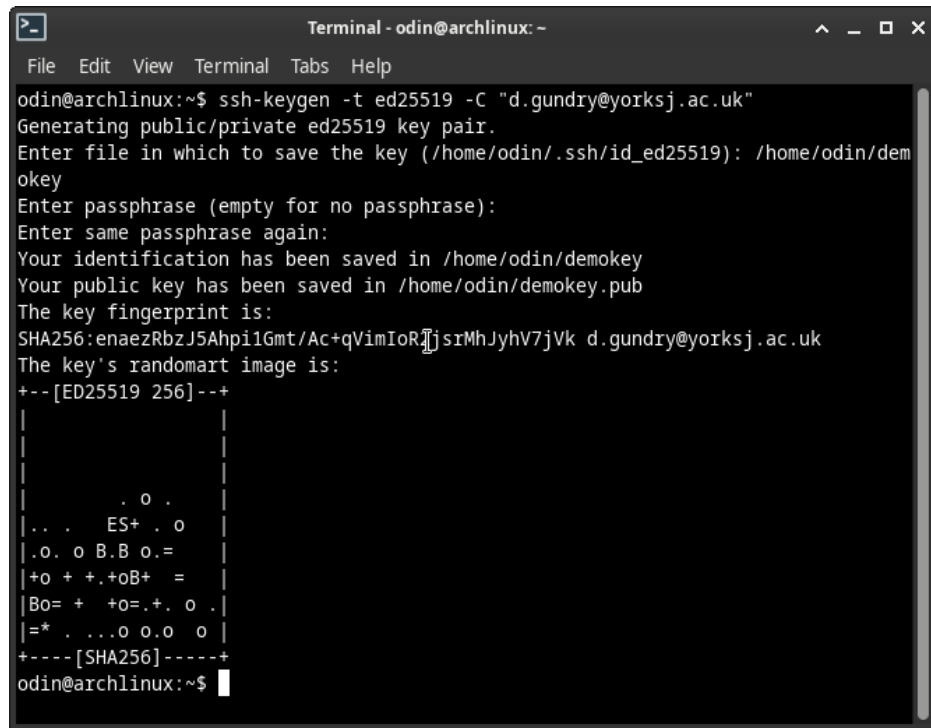


The background of the slide is a deep space image, likely the Hubble Deep Field, showing a dense field of galaxies and stars in shades of blue, purple, and red. A bright yellow rectangular box is positioned in the upper left quadrant, containing the text 'SSH authentication with SSH key' in black font.

SSH authentication with SSH key

- Another method is to create an ssh key

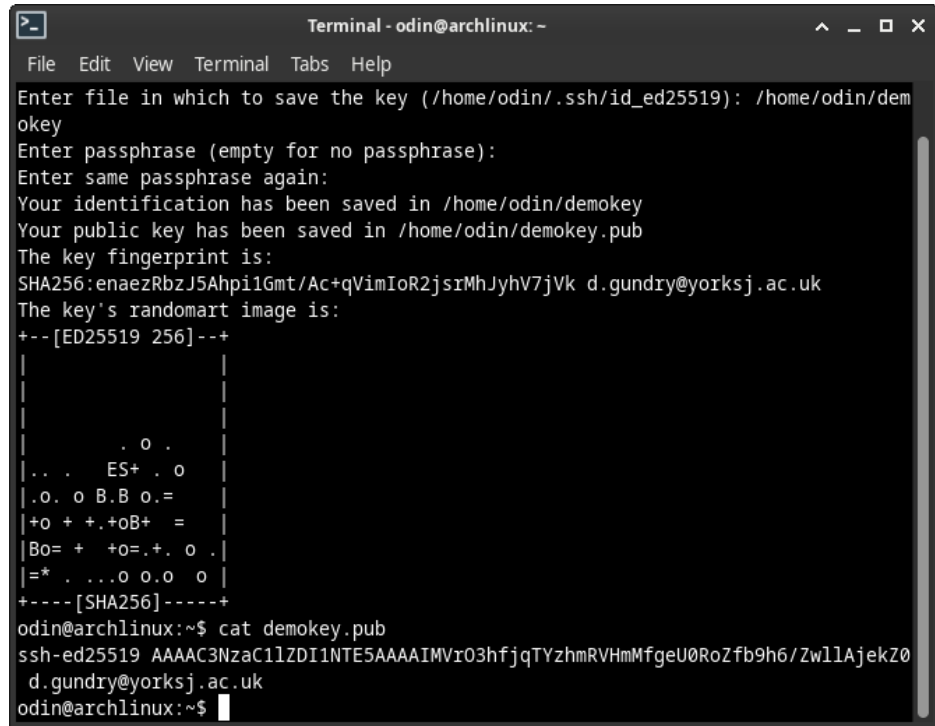
- Generate your ssh key
  - `ssh-keygen -t ed25519 -C`  
“<your email>”



```
Terminal - odin@archlinux: ~
File Edit View Terminal Tabs Help
odin@archlinux:~$ ssh-keygen -t ed25519 -C "d.gundry@yorks.j.ac.uk"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/odin/.ssh/id_ed25519): /home/odin/demokey
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/odin/demokey
Your public key has been saved in /home/odin/demokey.pub
The key fingerprint is:
SHA256:enaezRbzJ5Ahpil1Gmt/Ac+qVimIoRjjsrMhJyhV7jVvk d.gundry@yorks.j.ac.uk
The key's randomart image is:
+--[ED25519 256]--+
|
|      . O .
|.. .  ES+ . O
|.O. O B.B O.=
|+O + +.+OB+ =
|Bo= + +O=.+. O .
|=* . ...O O.O O
+-----[SHA256]-----+
odin@archlinux:~$
```



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

A terminal window titled "Terminal - odin@archlinux: ~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows the process of generating an SSH key pair. It prompts for a file name to save the key, a passphrase, and confirmation of the passphrase. It then displays the key fingerprint and a randomart image. Finally, it shows the command to cat the public key file and its contents.

```
Terminal - odin@archlinux: ~
File Edit View Terminal Tabs Help
Enter file in which to save the key (/home/odin/.ssh/id_ed25519): /home/odin/demokey
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/odin/demokey
Your public key has been saved in /home/odin/demokey.pub
The key fingerprint is:
SHA256:enaezRbzJ5Ahpil1Gmt/Ac+qVimIoR2jsrMhJyhV7jVkJ d.gundry@yorks.ac.uk
The key's randomart image is:
+--[ED25519 256]--+
|
|      . O .
|.. . ES+ . O
|.O. O B.B O.=
|+O + +. +OB+ =
|Bo= + +O=.+. O .|
|=* . ...O O.O O |
+----[SHA256]-----+
odin@archlinux:~$ cat demokey.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIMVrO3hfjqTYzhmRVHmMfgeU0RoZfb9h6/Zw1lAjekZ0
d.gundry@yorks.ac.uk
odin@archlinux:~$
```

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

The screenshot shows the GitHub Settings interface for a user named David Gundry. On the left is a sidebar with navigation links: Public profile, Account, Appearance, Accessibility, Notifications, Access, Billing and plans, Emails, Password and authentication, SSH and GPG keys (which is highlighted with a blue bar), Organizations, and Moderation. The main content area is titled 'SSH keys' and includes a 'New SSH key' button in the top right. Below the title is a message: 'This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.' A single SSH key is listed, represented by a green key icon, a blurred key string, and a 'Delete' button. The key was 'Added on 29 Jan 2021' and is 'Last used within the last week' with 'Read/write' permissions. Below the key list is a link to a guide on generating SSH keys. The 'GPG keys' section below it shows a message: 'There are no GPG keys associated with your account.' and a link to learn how to generate a GPG key. A 'New GPG key' button is in the top right of this section.

David Gundry  
Your personal account


Public profile  
Account  
Appearance  
Accessibility  
Notifications

Access  
Billing and plans  
Emails  
Password and authentication  
**SSH and GPG keys**  
Organizations  
Moderation

### SSH keys

New SSH key

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

 [blurred key string]  
SSH Added on 29 Jan 2021  
Last used within the last week — Read/write  
Delete

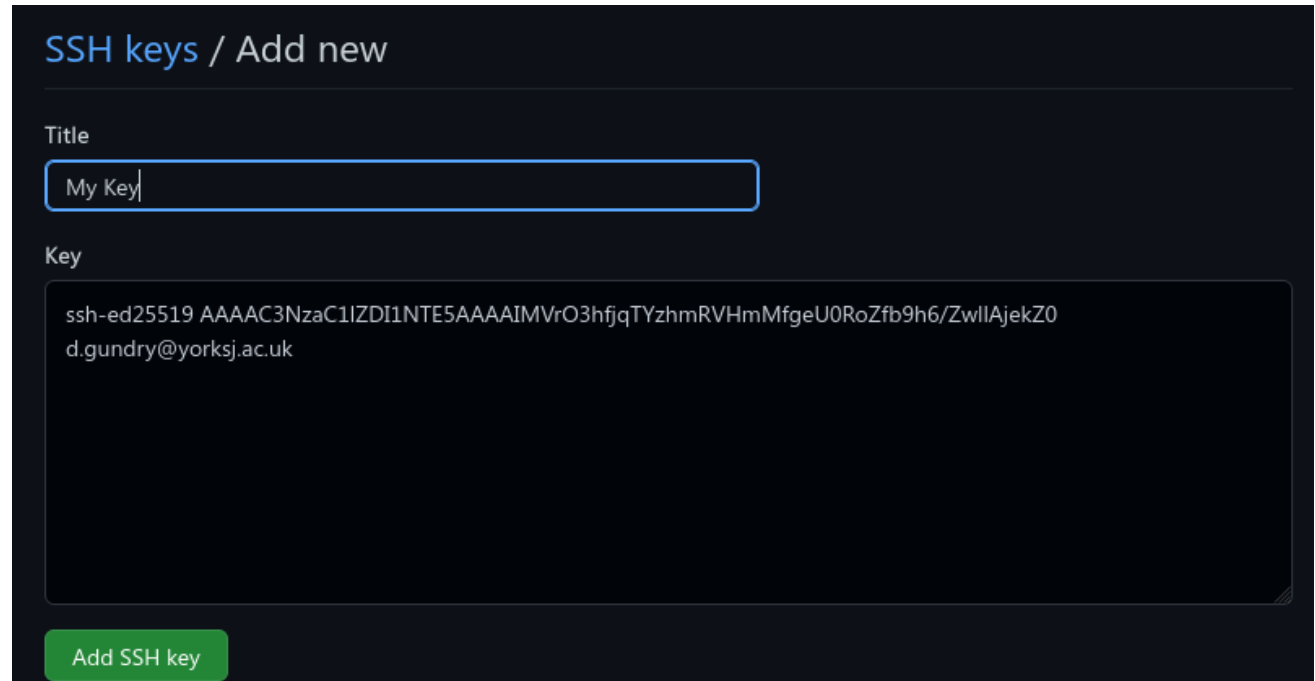
Check out our guide to [generating SSH keys](#) or [troubleshoot common SSH problems](#).

### GPG keys

New GPG key

There are no GPG keys associated with your account.  
[Learn how to generate a GPG key and add it to your account.](#)

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



The screenshot shows the 'SSH keys / Add new' page on GitHub. It features a dark-themed interface. At the top, the breadcrumb 'SSH keys / Add new' is displayed. Below this, there are two main input sections. The first section is labeled 'Title' and contains a text input field with the value 'My Key'. The second section is labeled 'Key' and contains a large text area with the SSH public key content: 'ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIMVrO3hfjqTYzhmRVHmMfgeU0RoZfb9h6/ZwIIAjekZO d.gundry@yorks.ac.uk'. At the bottom of the form, there is a green button labeled 'Add SSH key'.

SSH keys / Add new

Title

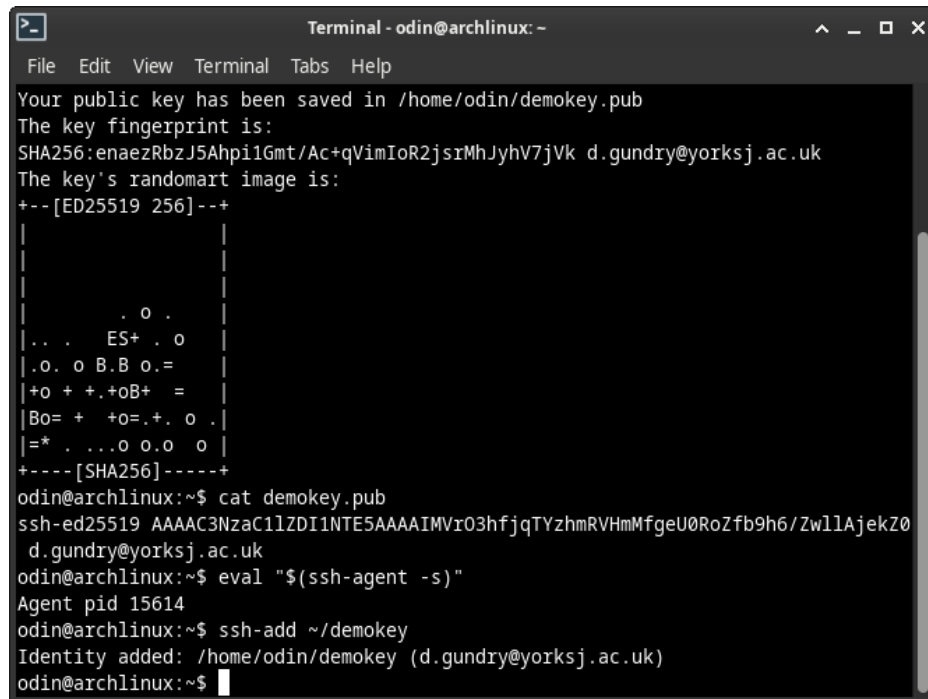
My Key

Key

```
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIMVrO3hfjqTYzhmRVHmMfgeU0RoZfb9h6/ZwIIAjekZO
d.gundry@yorks.ac.uk
```

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
- It will remember your key but only for the session



```
Terminal - odin@archlinux: -
File Edit View Terminal Tabs Help
Your public key has been saved in /home/odin/demokey.pub
The key fingerprint is:
SHA256:enaezRbzJ5AhpilGmt/Ac+qVimIoR2jsrMhJyhV7jVvk d.gundry@yorks.ac.uk
The key's randomart image is:
+--[ED25519 256]--+
|
|      . O .
|... .. ES+ . o
|.o. o B.B o.=
|+o + +. +oB+ =
|Bo= + +o=.+. o .
|=* . ...o o.o o |
+-----[SHA256]-----+
odin@archlinux:~$ cat demokey.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIMVrO3hfjqTYzhmRVHmMfgeU0RoZfb9h6/ZwllAjekZ0
d.gundry@yorks.ac.uk
odin@archlinux:~$ eval "$(ssh-agent -s)"
Agent pid 15614
odin@archlinux:~$ ssh-add ~/demokey
Identity added: /home/odin/demokey (d.gundry@yorks.ac.uk)
odin@archlinux:~$
```

- 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

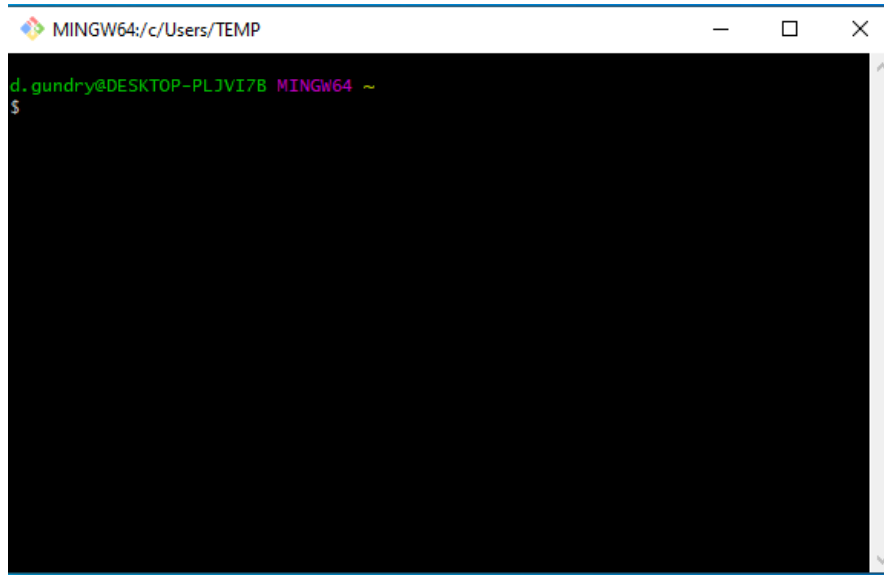




Create a local repository

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

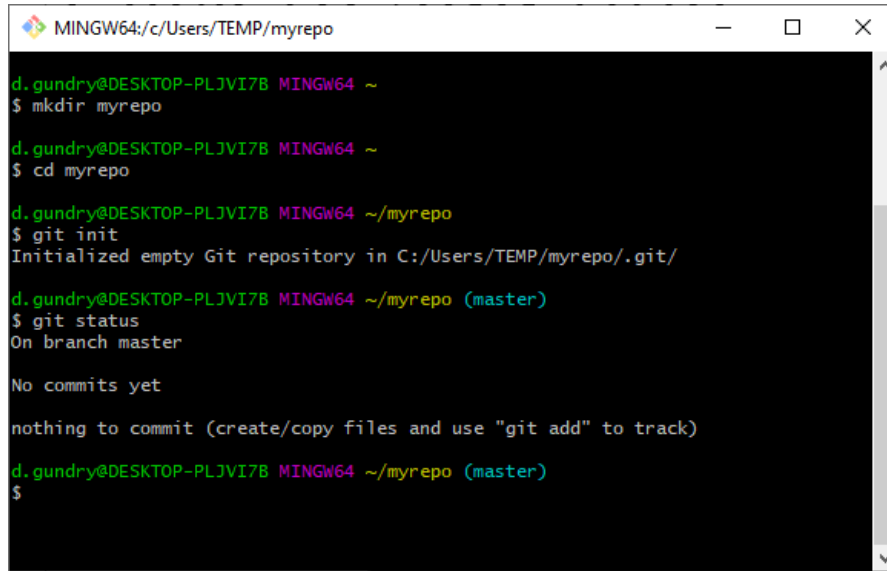
## Create a repository



- 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



```
MINGW64; c:/Users/TEMP/myrepo

d.gundry@DESKTOP-PLJVI7B MINGW64 ~
$ mkdir myrepo

d.gundry@DESKTOP-PLJVI7B MINGW64 ~
$ cd myrepo

d.gundry@DESKTOP-PLJVI7B MINGW64 ~/myrepo
$ git init
Initialized empty Git repository in C:/Users/TEMP/myrepo/.git/

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

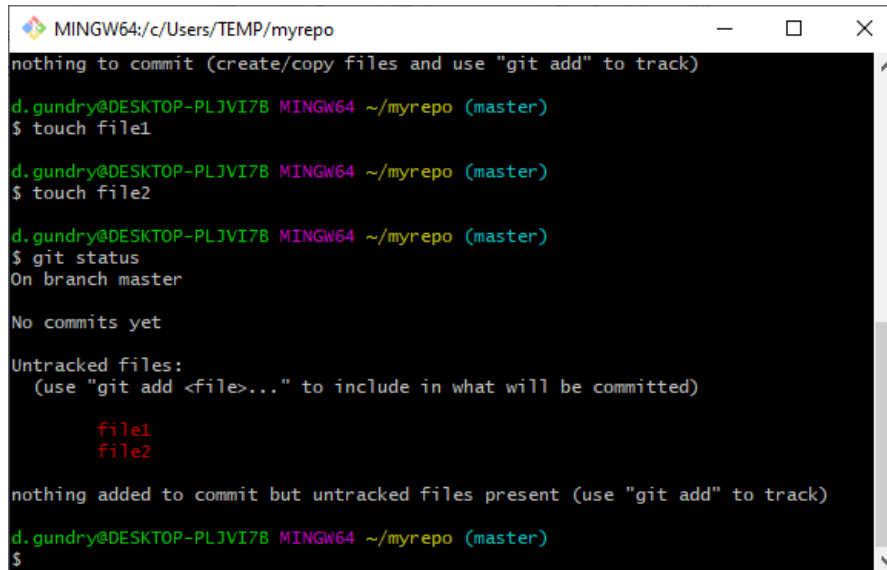
No commits yet

nothing to commit (create/copy files and use "git add" to track)

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

- 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**

## Create a repository



```
MINGW64; c/Users/TEMP/myrepo
nothing to commit (create/copy files and use "git add" to track)

d.gundry@DESKTOP-PLJVI7B MINGW64 ~/myrepo (master)
$ touch file1

d.gundry@DESKTOP-PLJVI7B MINGW64 ~/myrepo (master)
$ touch file2

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

No commits yet

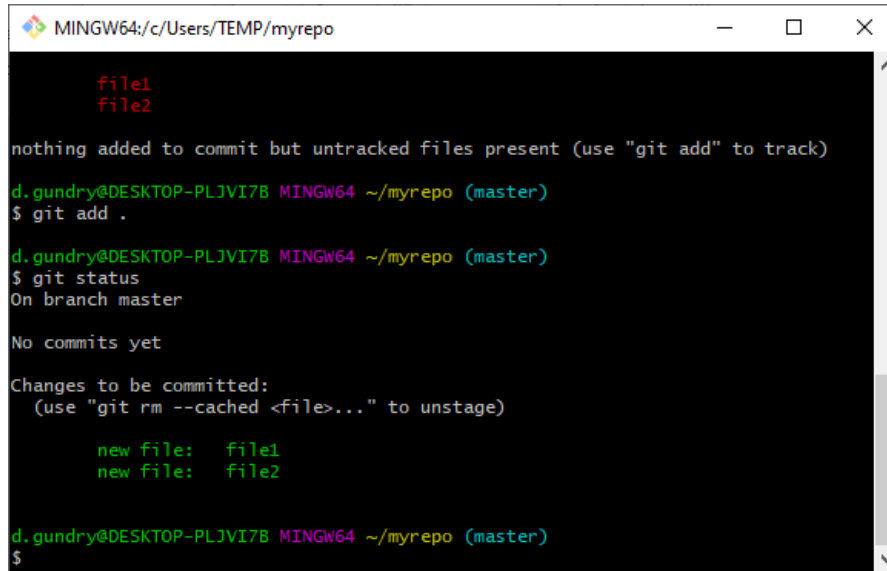
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        file1
        file2

nothing added to commit but untracked files present (use "git add" to track)
d.gundry@DESKTOP-PLJVI7B MINGW64 ~/myrepo (master)
$
```

- 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)

## Create a repository

A terminal window titled 'MINGW64; c/Users/TEMP/myrepo' with standard window controls. The terminal has a black background with red and green text. It shows the execution of 'git add .' and 'git status' commands. The output indicates that 'file1' and 'file2' are new files to be committed. The prompt is 'd.gundry@DESKTOP-PLJVI7B MINGW64 ~/myrepo (master)'.

```
MINGW64; c/Users/TEMP/myrepo

file1
file2

nothing added to commit but untracked files present (use "git add" to track)

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

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

No commits yet

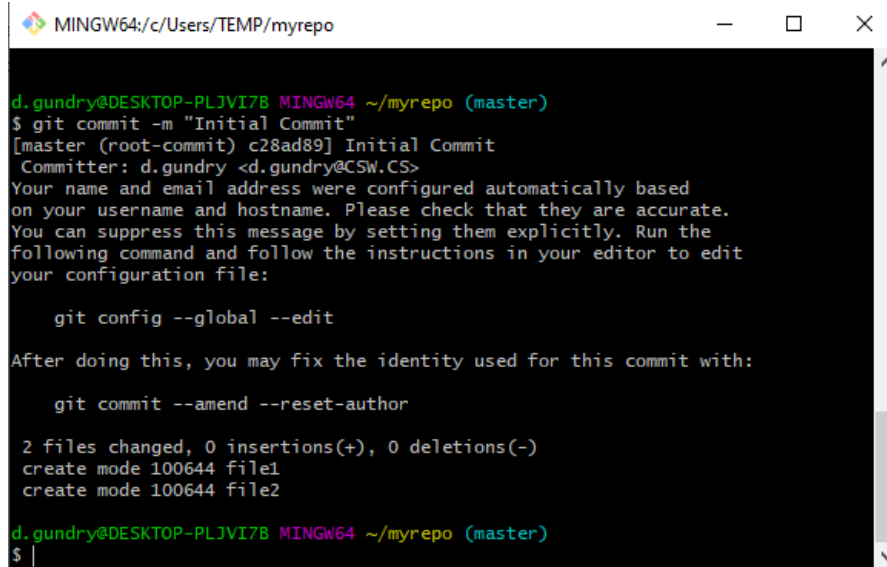
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   file1
        new file:   file2

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

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

## Create a repository



```
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:

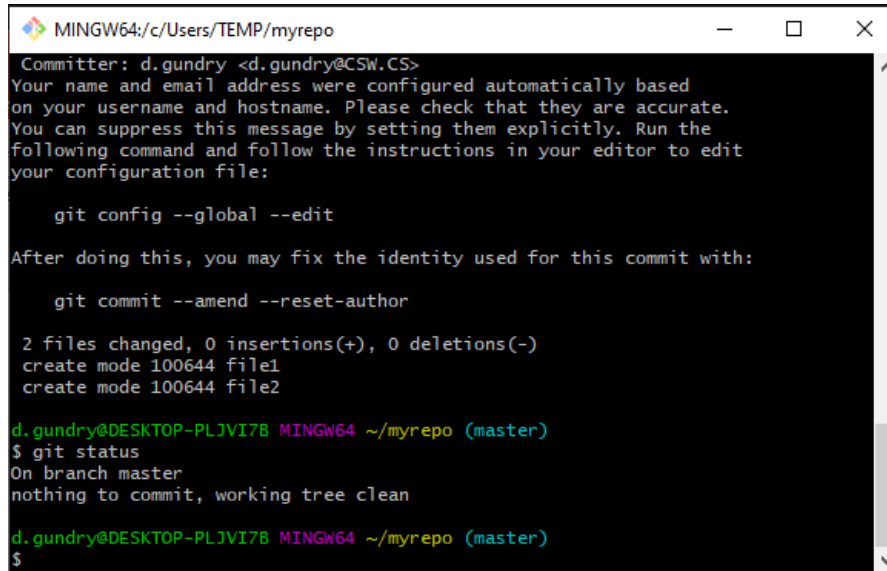
    git commit --amend --reset-author

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.

## Create a repository



```
MINGW64; c/Users/TEMP/myrepo
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)
$ git status
On branch master
nothing to commit, working tree clean

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

- 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





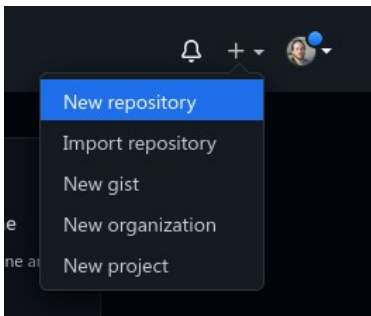
Set a remote and push commits

- 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

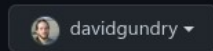


## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)

Owner \*



Repository name \*



Great repository names are short and memorable. Need inspiration? How about [reimagined-carnival?](#)

Description (optional)



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

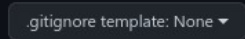


Add a README file

This is where you can write a long description for your project. [Learn more.](#)

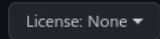
Add .gitignore


Choose which files not to track from a list of templates. [Learn more.](#)



Choose a license

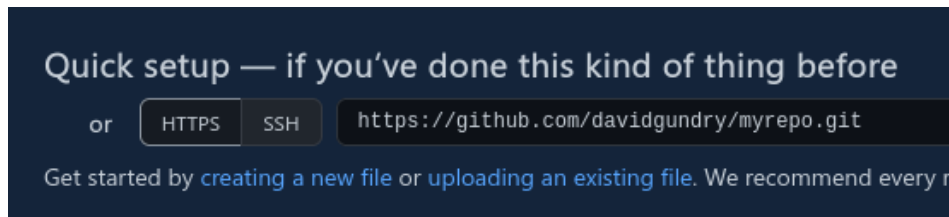
A license tells others what they can and can't do with your code. [Learn more.](#)



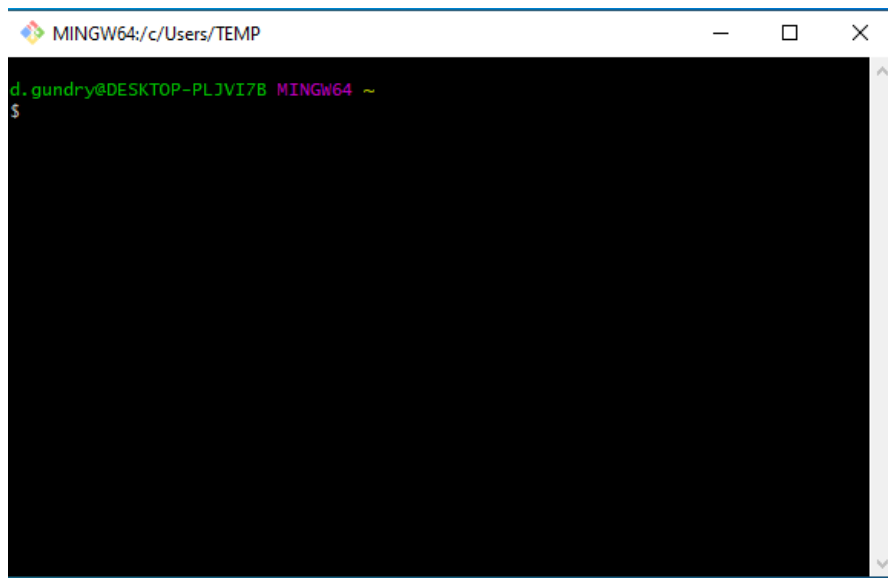
 You are creating a public repository in your personal account.

Create repository

- 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

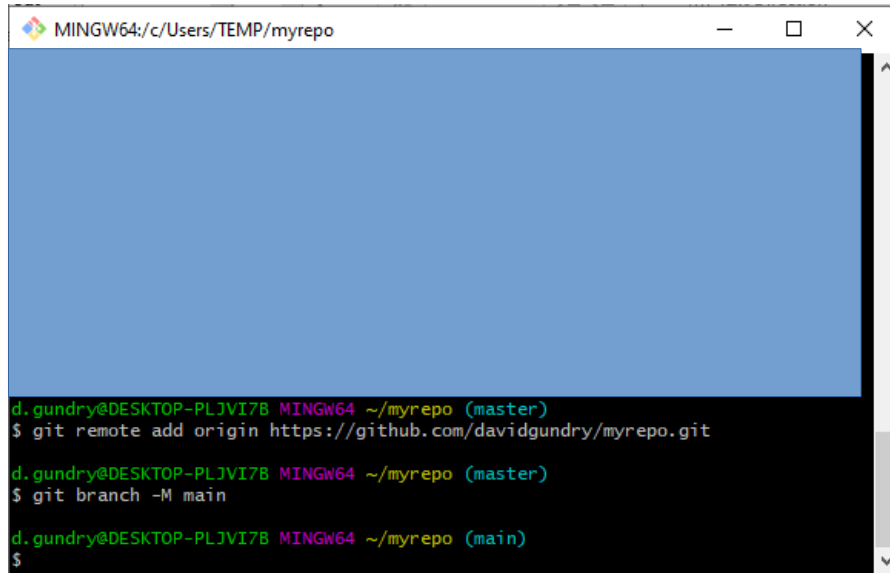


## Set remote and push commits



- 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

A screenshot of a Windows terminal window titled 'MINGW64; c/Users/TEMP/myrepo'. The terminal shows three lines of commands and their output. The first line shows the user adding a remote named 'origin' with the URL 'https://github.com/davidgundry/myrepo.git'. The second line shows the user renaming the current branch 'master' to 'main'. The third line shows the user's prompt after the branch has been renamed.

```
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)
$
```

- 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

## Set remote and push commits

- `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