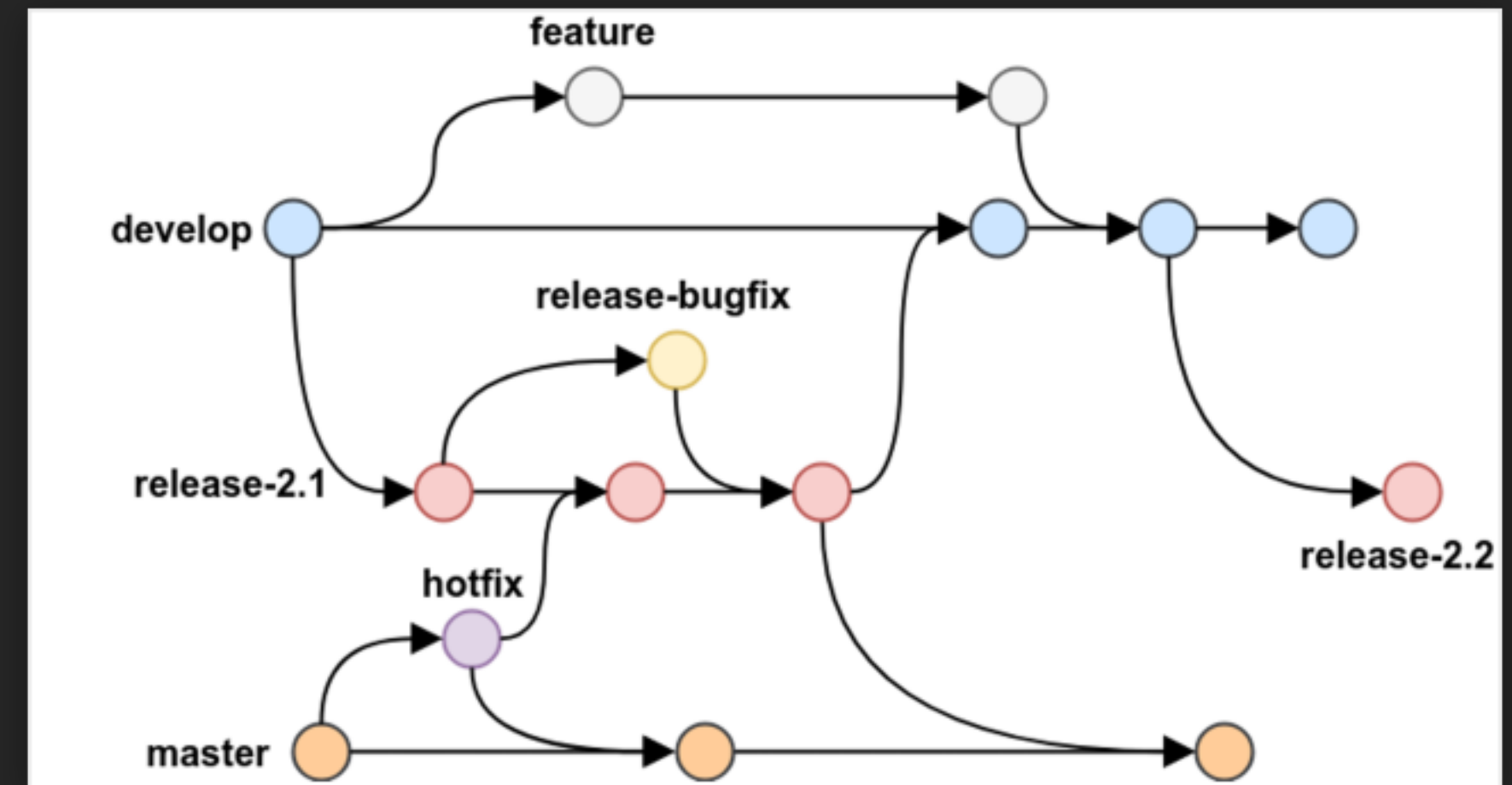


# Version Control Systems

```
module = Module(  
    code="ELEE1149",  
    name="Software Engineering",  
    credits=15,  
    module_leader="Seb Blair BEng(H) PGCAP MIET MIHEEM FHEA"  
)
```

# What are VCS?

- Tracking and managing changes:
- Work faster and more reliably
- Keeps track of all code modifications:
- Specialised Database (Repository)
- Solves Common Team Problems:
- Conflicting concurrent work,
- incompatibles due to concurrent working,
- having unstable releases



## Common Benefits of VCS

**Historical information** : Looking at the history of changes it is a lot easier to find where bugs have originated. Also, it might be easier to find the right team member best suited to fix an error.

**Branching**: Working concurrently on multiple issues, without interference. Working on different types of releases.

**Merging** : Making sure that team members work does not interfere with each other.

**Traceability** : Team members work more fluently together.

**Testing and Documentation**: Comments for each change and its association help producing better documentation. Creating tests is easier.

# Types of VCS

- **Local:**

- Creates a database on the your hardware

- **Centralised:**

- History of changes kept in a single database in a central server.
- Clients need to constantly communicate with the database and receive a partial working copy.

- **Distributed:**

- Single database in a central server that is also distributed among all clients
- Each client has a full working copy of the repository

# VCS – List

## Local Data Model

- Revision Control System (RCS) [OS]
- Source Code Control System (SCCS) [OS]
- The Librarian [P]
- Panvalet [P]

## Client Server model

- Concurrent Versions System [OS]
- Subversion (SVN) [OS]
- Vesta [OS]
- AccuRev [P]
- Autodesk Vault [P]
- CADES [P]
- ...
- Vault [P]

- Visual Source Safe [P]



# VCS - List

## Distrubted Model

- ArX [OS]
- Bazaar [OS]
- BitKeeper [OS]
- Fossil [OS]
- Git [OS]
- GNU Arch [OS]
- Mercurial [OS]
- Code Co-op [P]
- Sun WorkShop TeamWare [P]
- Plastic SCM [P]



# Git



- Developed by Linus Torvalds (you know the inventor/creator of Linux) to manage the development of Linux
- Over 76k commits to date (2025)
- First ever commit - `e83c5163316f89bfbde7d9ab23ca2e25604af290` (21 years ago)
- Commit message: `Initial revision of "git", the information manager from hell`

GIT - the stupid content tracker

"git" can mean anything, depending on your mood.

- random three-letter combination that is pronounceable, and not actually used by any common UNIX command. The fact that it is a mispronunciation of "get" may or may not be relevant.
- stupid. contemptible and despicable. simple. Take your pick from the dictionary of slang.
- "global information tracker": you're in a good mood, and it actually works for you. Angels sing, and a light suddenly fills the room.
- "goddamn idiotic truckload of sh\*t": when it breaks

...



# Characteristics

- A very popular VCS
- Performance
- Better performance compared to competitors
- Deals with the data in the file rather than the file properties
- Security
- Designed to provide security
- Uses SHA-256 encryption
- Flexibility
- Non-linear development
- Detailed log of information



# Global Configuration

`git config` is a allows you to set important configuration settings, it's easy to get Git to work exactly the way you, your company, or your group needs it to.

Location of config:

```
$HOME\.gitconfig
```

```
~/.gitconfig
```

Set via CLI:

```
$ git config --global user.name "your user name"  
$ git config --global user.email "your email"
```

or modify the configuration file directly:

```
$ git config --global --edit
```

```
<nano/vim/vi/...> ~/.gitconfig
```

.gitconfig file

```
[user]  
    name = CompEng0001  
    email = s.blair@gre.ac.uk  
  
[core]  
    editor = vim  
    excludesFile = "~/.config/git/.gitignore"  
    attributesFile = "~/.config/git/.gitattributes"  
  
[grep]  
    lineNumber = true  
    patternType = perl
```

# Functions

All commands are prepended with `git`

- `init` initialise a folder as a git repository
- `fetch`, `pull`: Get a working copy of a repository
- `add`, `commit`, `push`: Record a change or changes in at least one of the files stored in the repository.
- `branch`: Create a copy of a repository to be worked independently.
- `merge`: Collates changes of two different copies of a repository.
- `log`: Records information of each change within a repository

```
> git --help
usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
          [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
          [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
          [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
          [--config-env=<name>=<envvar>] <command> [<args>]

These are common Git commands used in various situations:


start a working area (see also: git help tutorial)
  clone      Clone a repository into a new directory
  init       Create an empty Git repository or reinitialize an existing one


work on the current change (see also: git help everyday)
  add        Add file contents to the index
  mv         Move or rename a file, a directory, or a symlink
  restore    Restore working tree files
  rm         Remove files from the working tree and from the index


examine the history and state (see also: git help revisions)
  bisect     Use binary search to find the commit that introduced a bug
  diff       Show changes between commits, commit and working tree, etc
  grep       Print lines matching a pattern
  log        Show commit logs
  show       Show various types of objects
  status     Show the working tree status


grow, mark and tweak your common history
  branch     List, create, or delete branches
  commit     Record changes to the repository
  merge      Join two or more development histories together
```

# Structure of the .git directory

```
.git/
├── objects/           # Git's object database
│   ├── pack/         # Compressed repository data
│   └── info/         # Additional object information
│       └── [sha1]/    # Object storage using hash prefixes
├── refs/             # Reference storage
│   ├── heads/        # Branch references
│   ├── tags/         # Tag references
│   └── remotes/      # Remote repository references
├── HEAD              # Points to current branch
├── config             # Repository configuration
├── index              # Staging area information
├── hooks/            # Scripts for automation
│   ├── pre-commit    # Run before commit is created
│   └── post-commit    # Run after commit is created
├── info/             # Repository information
│   └── exclude        # Local ignore patterns
└── logs/             # Reference history
    ├── HEAD          # History of HEAD updates
    └── refs/         # Branch update history
```

# Example

```

Git\tmp\myawesomerepo
> git init
Initialized empty Git repository in C:/Users/dev/Git/tmp/myawesomerepo/.git/

myawesomerepo on ʘ main
> touch README.md

myawesomerepo on ʘ main [?]
> git status
On branch main

No commits yet

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

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

myawesomerepo on ʘ main [?]
> git add README.md

myawesomerepo on ʘ main [+]
> git status
On branch main

No commits yet

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

myawesomerepo on ʘ main [+]
> git commit -m "init: initial commit"
[main (root-commit) 43ba0aa] init: initial commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md

myawesomerepo on ʘ main
> git log
commit 43ba0aa4935549d2d4245bedf455bbc25a5d1636 (HEAD -> main)
Author: CompEng0001 <s.blair@gre.ac.uk>
Date:   Mon Apr 14 16:10:45 2025 +0100

    init: initial commit

myawesomerepo on ʘ main
> git show 43ba0aa
commit 43ba0aa4935549d2d4245bedf455bbc25a5d1636 (HEAD -> main)
Author: CompEng0001 <s.blair@gre.ac.uk>
Date:   Mon Apr 14 16:10:45 2025 +0100

    init: initial commit

diff --git a/README.md b/README.md
new file mode 100644
index 0000000..e69de29

```

# Repository Staging Area

- As part of the version control features there is the **Staging Area**.
- This is feature enables the developer to move files independently of each other to the repository.

```
git add <filename>
```

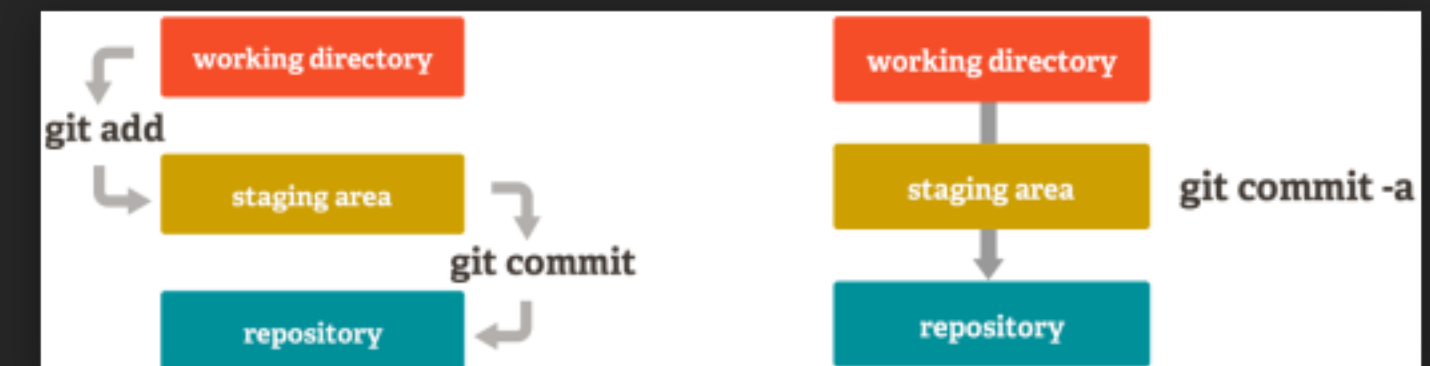
```
git add .
```

```
git commit
```

```
git commit -a
```

```
git commit -m
```

```
git commit -am
```



# Hashing

- Often referred to as a commit hash, is a unique identifier for every single commit in a Git repository
- SHA-1 cryptographic hash function
- 40-character string of hexadecimal characters (0-9 and a-f)
- Hashes are calculated based on the contents of the files in the commit, the metadata of the commit (like timestamp and author), and the parent commit's hash.

```
myawesomerepo on ʘ main  
> git log  
commit 43ba0aa4935549d2d4245bedf455bbc25a5d1636 (HEAD -> main)  
Author: CompEng0001 <s.blair@gre.ac.uk>  
Date:   Mon Apr 14 16:10:45 2025 +0100  
  
    init: initial commit
```

- Comparing hashes to see differences `git diff <a_hash> <another_hash>`

```
git diff 43ba0aa4935549d2d4245bedf455bbc25a5d1636 a1b2c3d4e5f67890abcdef1234567890abcdef12
```



# Collisions!

- No two hashes should ever be the same!
- A hash collision occurs when two different inputs generate the same output hash.
- Output length: **160 bits**

Possible hashes:

$$2^{160} = 1.4615016373309029182036848327163 \times 10^{48}$$

or if you prefer:

- $2^{160}$  grains of sand ( $0.5mm^3$ ) would fill 500 trillion Jupiters

The probability  $P(n)$  of at least one collision is approximately:

$n$ (number of hashes)	$P(n)$ (collision probability)
1	$\approx 0$
$1 \cdot 10^6$	$< 10^{-47}$
$1.4 \times 10^{24}$ (septillion)	$\approx 0.5$ (50%)

# Example of `sha1sum`

```
sha1sum.exe <<< hello  
f572d396fae9206628714fb2ce00f72e94f2258f *-
```

```
sha1sum.exe <<< Hello  
1d229271928d3f9e2bb0375bd6ce5db6c6d348d9 *-
```

```
myawesomerepo on ʘ main  
> cat README.md
```

```
myawesomerepo on ʘ main  
> sha1sum.exe README.md  
da39a3ee5e6b4b0d3255bfef95601890afd80709 *README.md
```

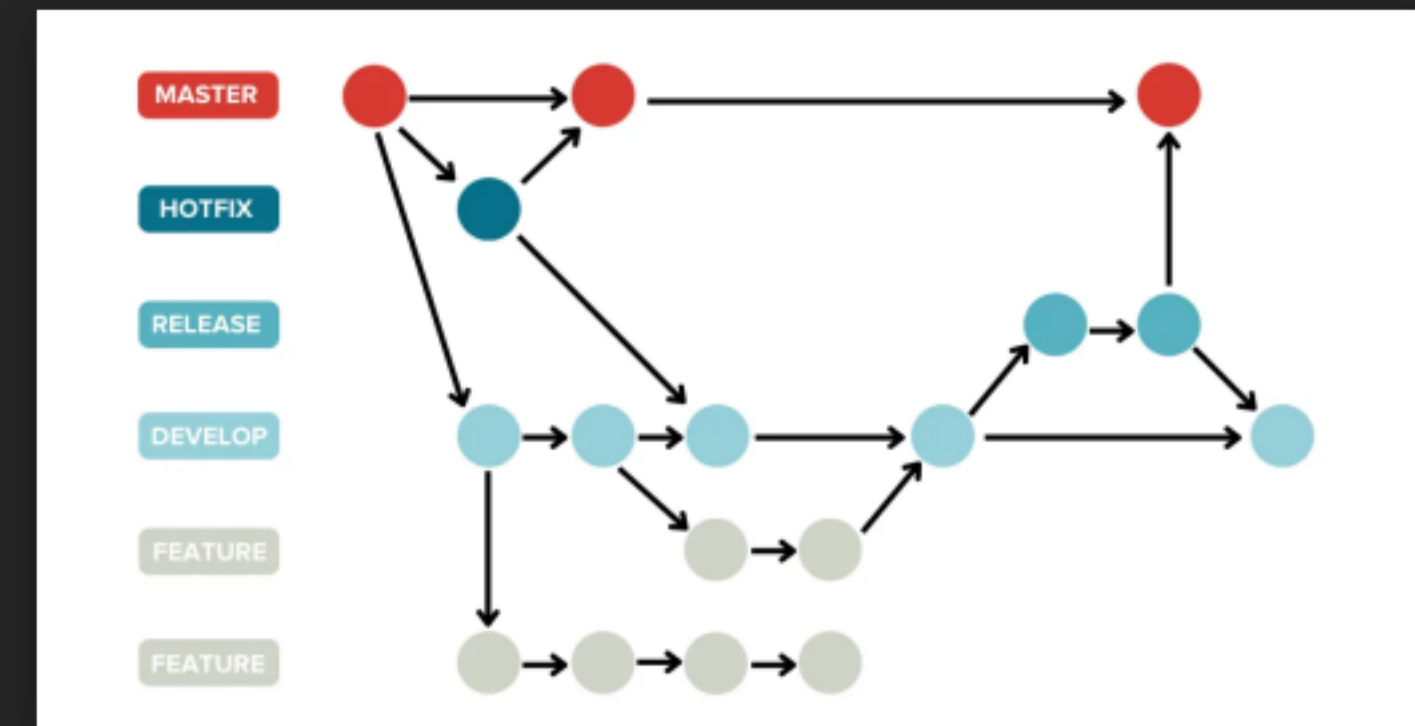
```
myawesomerepo on ʘ main  
> echo "# Introduction" > README.md
```

```
myawesomerepo on ʘ main [!]  
> sha1sum.exe README.md  
b1e4177033ba531626dbb54ae94b0b179e7aa0d3 *README.md
```

# Branching

# Branching

- **Master/Main:** The default development branch. Whenever you create a Git repository, a branch named "`master`" or "`main`" is created, and becomes the active branch.
- **HotFix:** To patch urgent issues found in the `master/main` branch (e.g., bugs in production)
- **Release:** To prepare a new production release
- **Develop:** This is another branch, which is a way to **edit/develop/test** code without changing the `main` branch.
- **Feature:** For developing new features or enhancements.



# Example

```
myawesomerepo on ̯ main
> git switch -c dev
Switched to a new branch 'dev'
```

```
myawesomerepo on ̯ dev
> touch config.toml
```

```
myawesomerepo on ̯ dev [?]
> git add config.toml
```

```
myawesomerepo on ̯ dev [+]
> git commit -m "add: a new empty config.toml"
[dev 9771640] add: a new empty config.toml
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 config.toml
```

```
myawesomerepo on ̯ dev
> git log
commit 9771640b540b8b6fcabf9c9fbfb74622924122100 (HEAD -> dev)
Author: CompEng0001 <s.blair@gre.ac.uk>
Date: Mon Apr 14 17:05:45 2025 +0100
```

```
    add: a new empty config.toml
```

```
commit 43ba0aa4935549d2d4245bedf455bbc25a5d1636 (main)
Author: CompEng0001 <s.blair@gre.ac.uk>
Date: Mon Apr 14 16:10:45 2025 +0100
```

```
    init: initial commit
```

```
myawesomerepo on ̯ dev
> git switch main
Switched to branch 'main'
```

```
myawesomerepo on ̯ main
> git log
commit 43ba0aa4935549d2d4245bedf455bbc25a5d1636 (HEAD -> main)
Author: CompEng0001 <s.blair@gre.ac.uk>
Date: Mon Apr 14 16:10:45 2025 +0100
```

```
    init: initial commit
```

```
myawesomerepo on ̯ main
> ls
README.md
```

# Merging & Rebasing

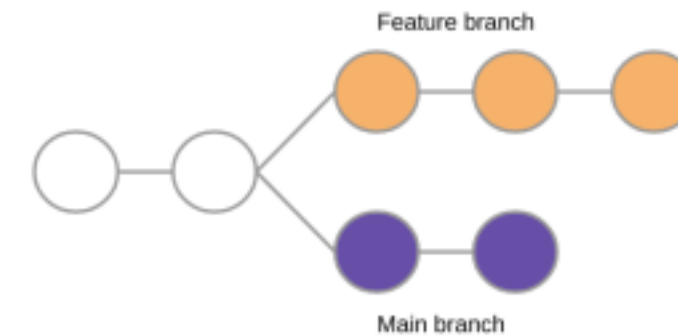
- merge

- A merge commit has two or more parent commits – one for each branch that was merged
- Merging happens at the commit level – Git merges entire commits together
- This keeps the history of that other feature branch in case you ever need it.

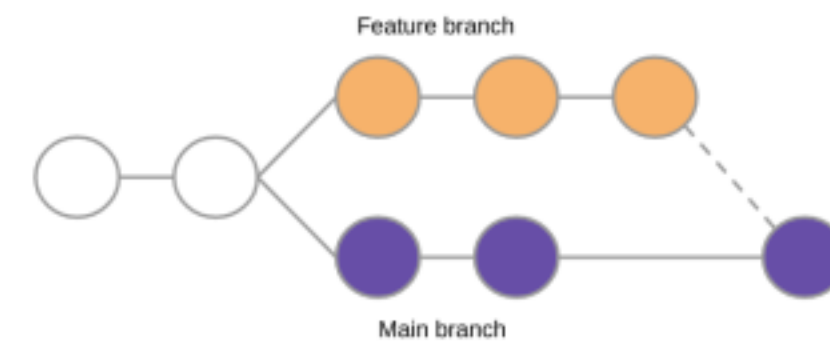
- rebase

- This will take the changes of your feature branch and append them to the main branch, which effectively removes the history as a separate branch of work.

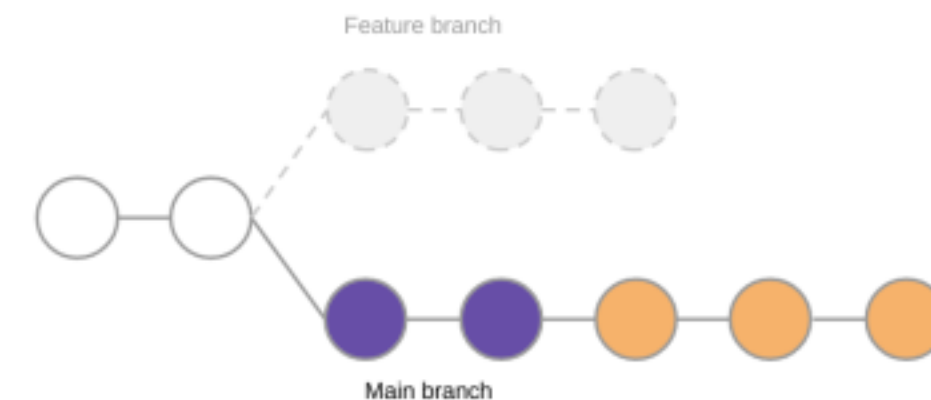
## Git Branch



## Git Merge



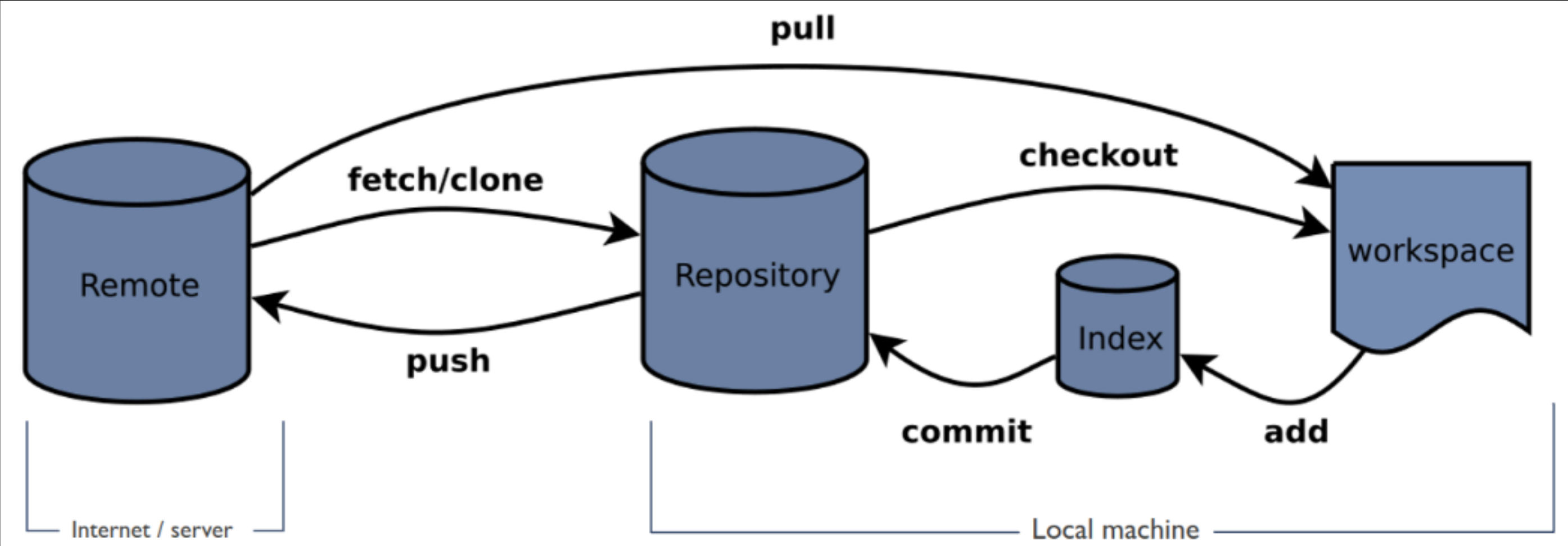
## Git Rebase



# Distribution



# Flow Diagram



# Hosts

# List

- TaraVault
- BitBucket
- SourceForg
- GitLab
- Gogs
- GitBucket
- GitHub
- AWS CodeCommit
- BeanStalk
- Phabricator
- ...many more

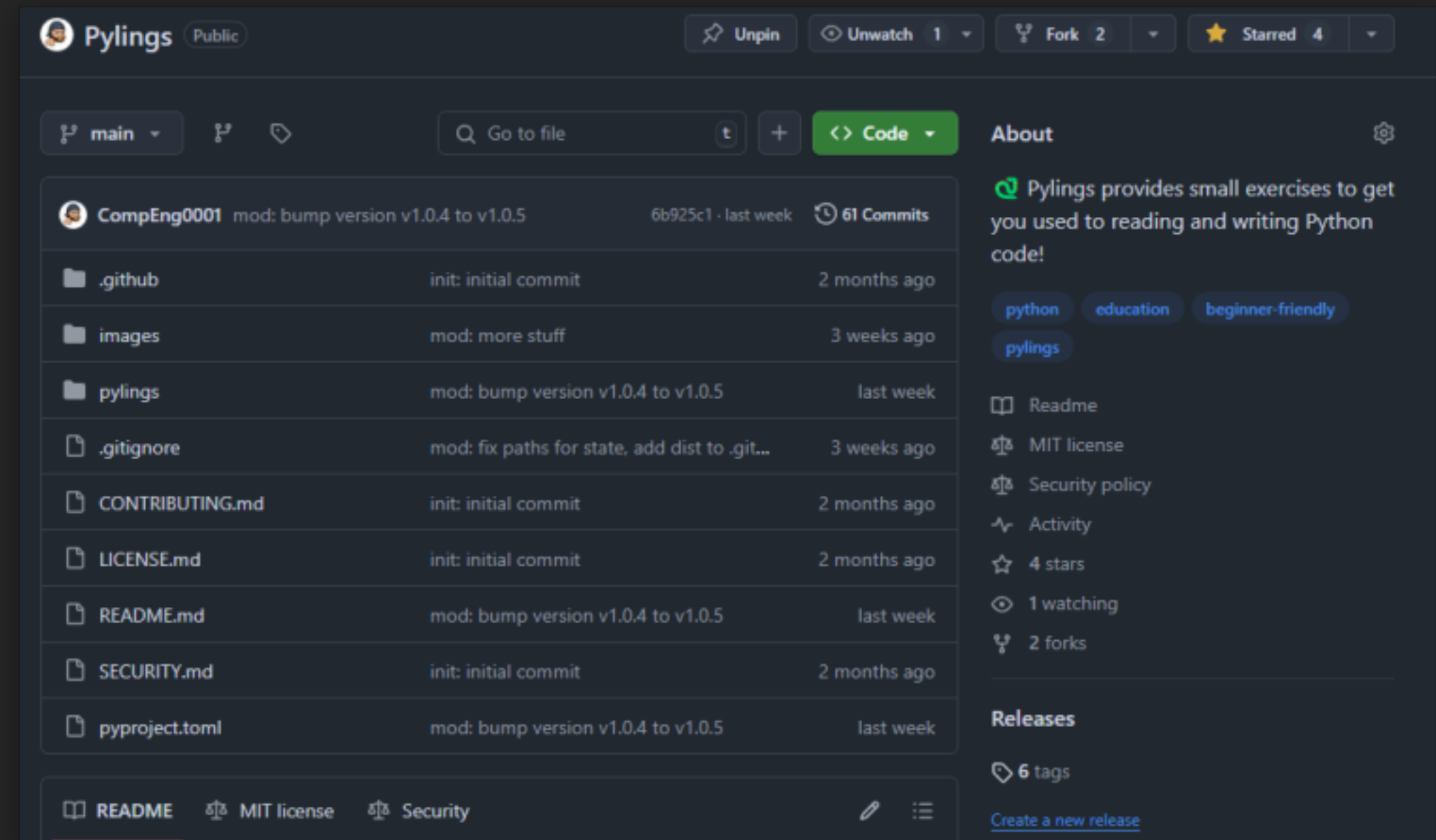


GitHub

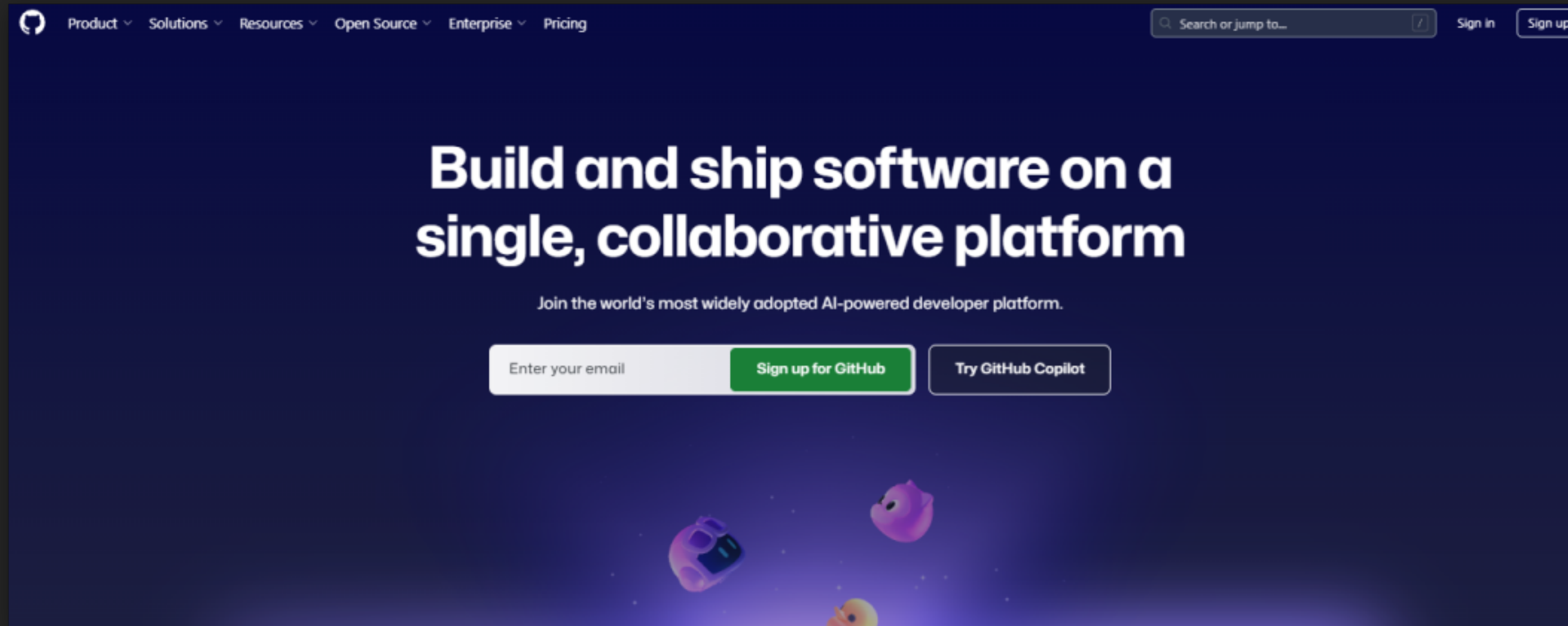


# GitHub – What is it and features?

- Web-based graphical user interface (GUI)
- Features
- Can act as a project manager
- Hosts Git repositories
- Secure with keys



# Account creation



\*Hint: use your `username@gre.ac.uk` (you can register for education package later)

# Encryption - ed25519

- The ED25519 key fingerprint is a unique identifier for an SSH key, ensuring secure connections between clients and servers. It is generated using the SHA-256 hashing algorithm.

```
$ ssh-keygen -t ed25519 -C "s.blair@gre.ac.uk"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/user/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_ed25519
Your public key has been saved in /home/user/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:Ue/4v8XYf6tK8DZ1zN8KHZl1k+GJZNmjqtz3KbR1nCo s.blair@gre.ac.uk
The key's randomart image is:
+--[ED25519 256]--+
|      .+.o=.      |
|      + B.Eo      |
|      o B.+       |
|      . + X       |
|      o S o       |
|      = .         |
|      . .         |
|      .           |
|                  |
+-----[SHA256]-----+
```

# Syncing Git and GitHub

```
/usr/bin/bash --login -i

> ssh-keygen.exe -t ed25519 -C "s.blair@gre.ac.uk"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/Sebastian Blair/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/Sebastian Blair/.ssh/id_ed25519
Your public key has been saved in /c/Users/Sebastian Blair/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:yDyCqOwbvr7moIXZjih58lHbiirmX7PP3E60LGA3b1U s.blair@gre.ac.uk
The key's randomart image is:
+---[ED25519 256]---+
|
| . . 0 .   E
|.. . o = S .
|o+ + * . .
|=+= +o+ ..
|@*oo.o*o=
|&#0...+*.+
+-----[SHA256]-----+
>
```

SSH keys / Add new

Title

Key type

Key

```
> ssh git@github.com
PTY allocation request failed on channel 0
Hi CompEng0001! You've successfully authenticated, but GitHub does not provide shell access.
Connection to github.com closed.
```



# Learning Git

