ITP20004 – Open-Source Software Labs

Project Management

Charmgil Hong

charmgil@handong.edu

Spring, 2023

Handong Global University

Announcements

- Weekly schedule

Week	Mon		Week	Thur	
1	Course overview, motivation, administrivia		1	CPR: C Programming Reinforcement - Functions	
2	Computer organization and Linux environment (1)		2	CPR: C Programming Reinforcement - Strings	
3	Computer organization and Linux environment (2)		3	CPR: C Programming Reinforcement - User-defined types, and memory allocation	
4	Basic Linux commands + Writing code on Linux (vim)		4	Getting started with Linux / Hands-on Linux command-line tools	
5	More Linux commands		5	CPR: C Programming Reinforcement - Understanding compilation and build process	
6	Project management (1)	**Proj 1 출제**	6	Project management (2)	
7	-		7	Project: BASIC interpreter	Project 1
8	-	Tuesday night: Midterm exar	8	Project QnA (Optional)	
9	CPR: C Programming Reinforcement - Accessing files and		9	Debugging with GDB + Unit testing with gtest	**Project 1 due: Week#9 Saturday**
10	Shell script			Shell script	
11	Code review GNU utilities	**Proj 2 출제**	12	Writing an application in C	
12	Github and open-source community			Using Github	**Project 2 due: Week#12 Saturday**
13	Computer network basics		10	Linux network commands	
14	Project: Text-based Game		14	Socket programming	
15	Project: Multi-user game	**Proj 3 출제**	15	Project: Multi-user game	
16	Final exam		16		**Project 3 due: Week#16 Saturday**

Announcements

- For each lab
  - Before a lab, every student submits a pre-lab report (worksheet-type assignment) – **individual work**
  - After a lab, each team sees and reports to the TA with the results – **team work**

- Up-coming schedule
  - There are no post-lab session on May 16 (Week #12)
  - We DO have a post-lab session on May 23 (Week #13)

Last Lab: File Permissions

- Permissions

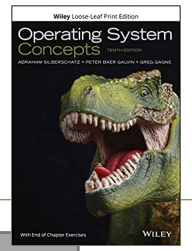
```
charmgil@vm-VirtualBox:~/private/scratch/lab3$ ls -l
total 488
-rw-r--r-- 1 charmgil charmgil  74556 Mar 18 21:52 CRICKET.WAV
-rw-r--r-- 1 charmgil charmgil  31730 Mar 18 21:50 hgu_logo.jpg
-rw-r--r-- 1 charmgil charmgil   1440 Mar 10 17:16 kernel5.txt
-rw-r--r-- 1 charmgil charmgil 217993 Mar 18 23:08 lab3data.zip
-rw-r--r-- 1 charmgil charmgil  47207 Mar 18 21:50 linux_PNG1.png
-rwxr-xr-x 1 charmgil charmgil 114496 May 14  2018 x64-intro.pdf
```

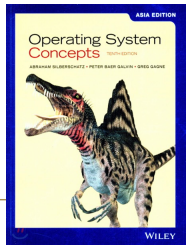
File System

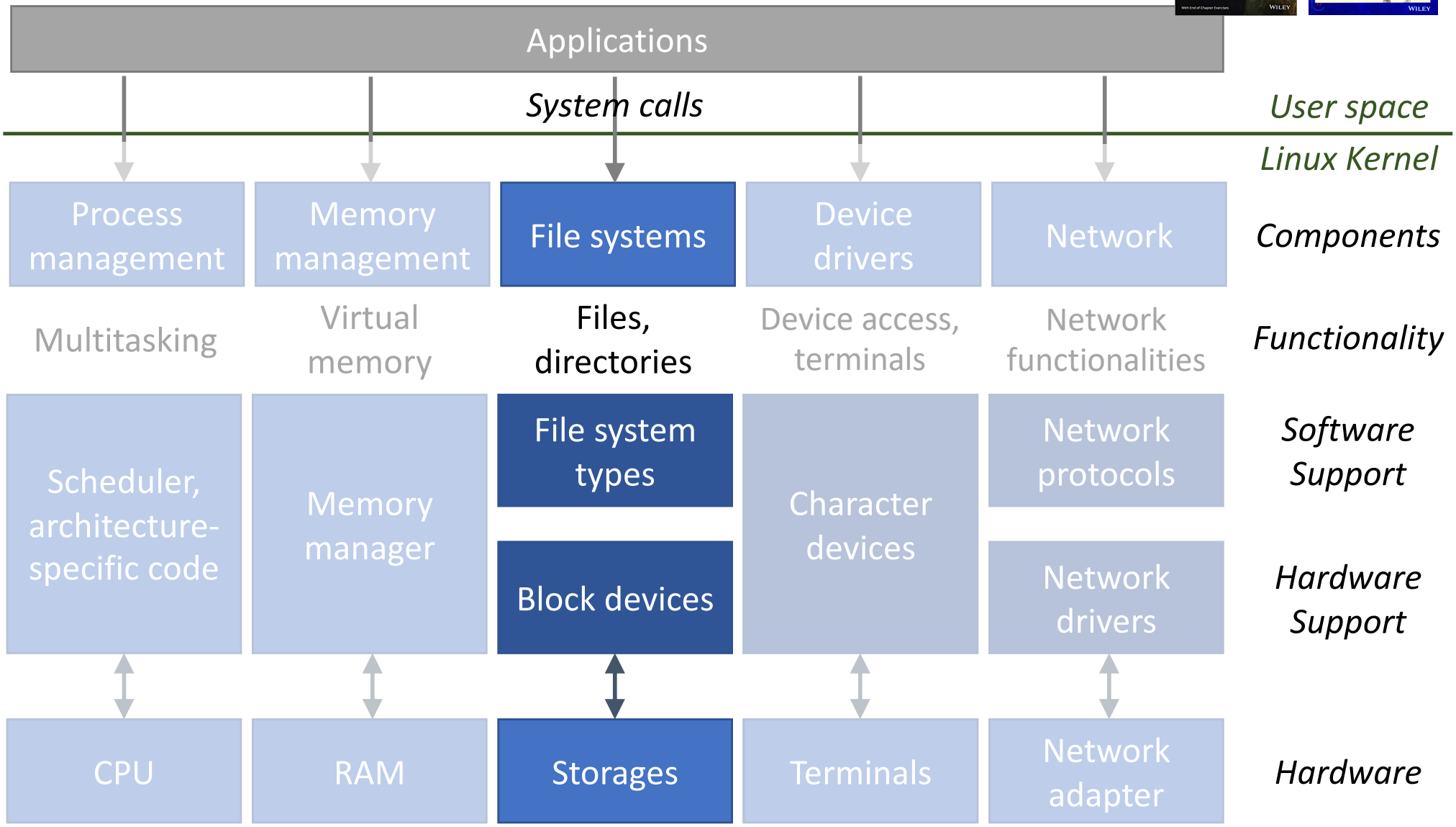
- A file is a logical unit of storage for a sequence of records
  - A storage is recording medium that holds information
  - A file must have a unique name (identifier) in its directory
  - Consists of one or more streams of bytes and a set of attributes (properties)

- A file system is the methods and data structures that an OS uses to keep track of files on storage devices
  - Provides a way to separate data into pieces (files and directories)
    - *C.f.*, Linux kernel treats every I/O device as a file (/dev directory) – Linux simplifies the I/O communications by making (abstracting) them as file accesses

Linux Kernel







History

- Electronic Recording Machine Accounting (ERMA) Mark 1 (1958)
- Microsoft's FAT (8-bit) (1977)
- Apple's DOS 3.x (1978)
- Rémy Card's ext, ext2 (1992, 1993)

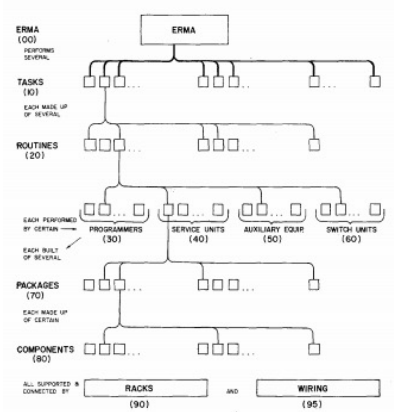


Fig. 2. The ERMA Mark I hierarchical structure

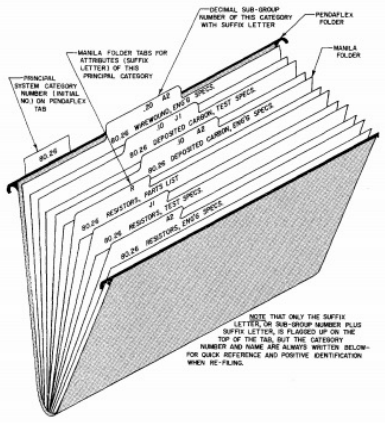
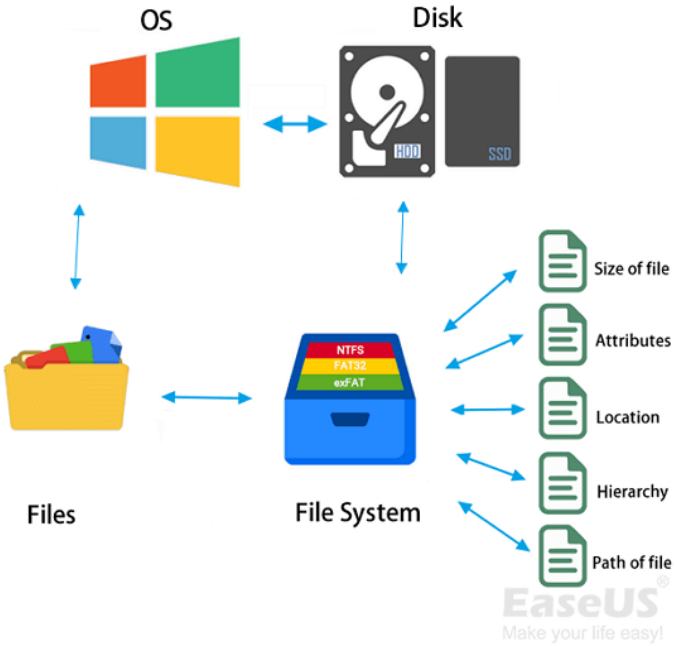
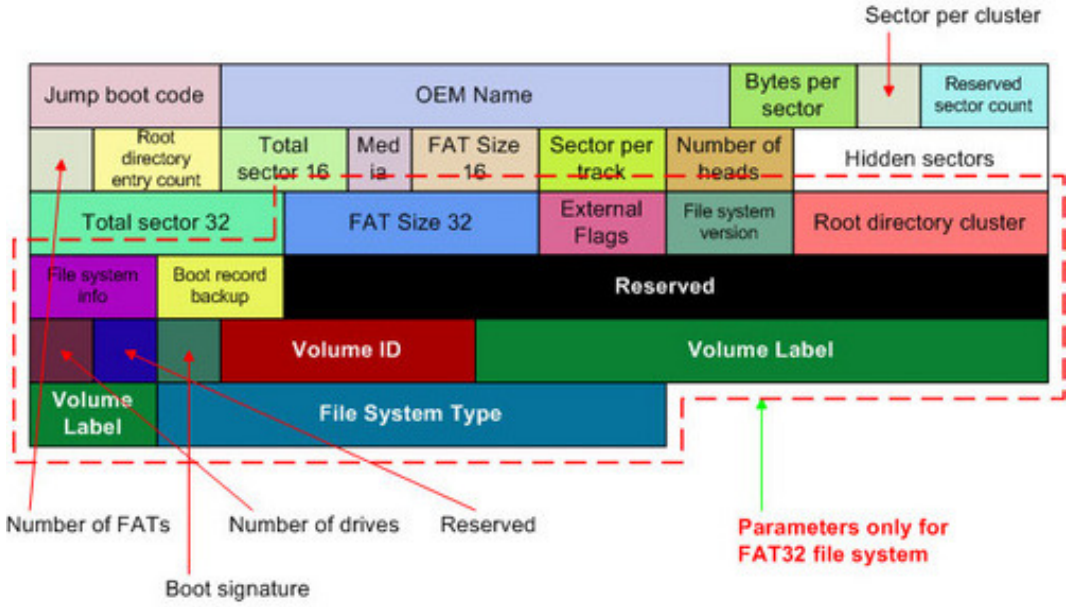


Fig. 3. Records storage and arrangement

File Systems Galore

- ext4 (Linux; 2006)
- APFS (Apple; 2016)
- NTFS (Microsoft; 1993)
- exFAT (Microsoft; 2006)), FAT32 (Microsoft; 1996)
- ZFS (Sun; since 2004)



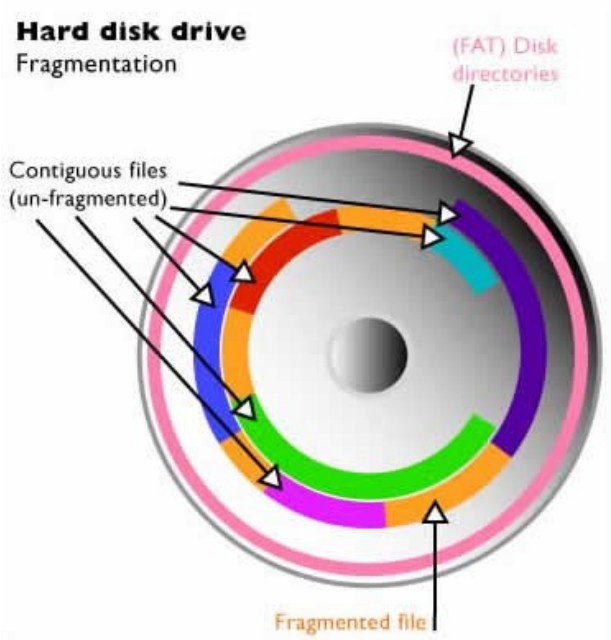


The items of FAT32 boot record

by **iprinceps**, <iprinceps@gmail.com>

File Fragment

- FAT*, NTFS (with older versions of Windows)



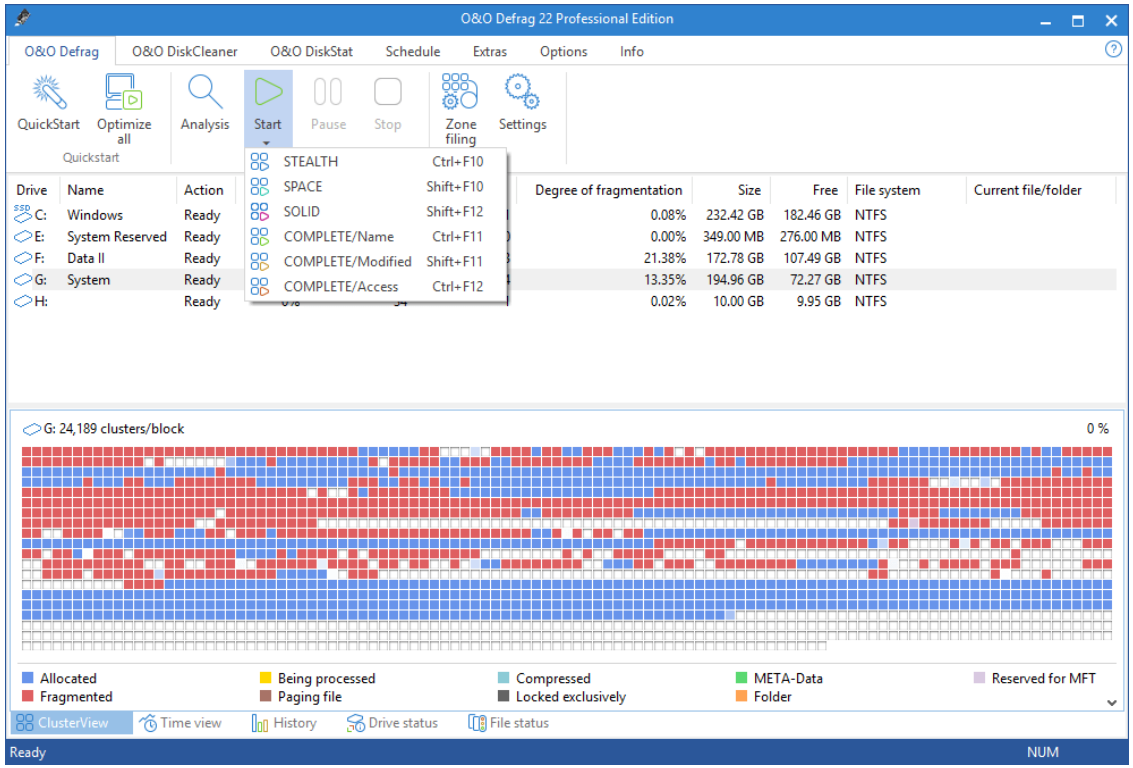


*Image obtained from http://www.planetoftunes.com/computer/hard-disc-drives.html#.XJP4hBMzaL4

Files on Linux

- A file and its metadata
  - Metadata: OS allocates a few bytes of storage to carry some basic information about each file

- `stat` – Display file status

```
charmgil@vm-VirtualBox:~/private/scratch$ stat hello.c
  File: hello.c
  Size: 90              Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d      Inode: 552269      Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/charmgil)   Gid: ( 1000/charmgil)
Access: 2019-03-18 21:28:08.105502374 +0900
Modify: 2019-03-05 04:02:54.566528235 +0900
Change: 2019-03-05 04:02:54.570530236 +0900
 Birth: -
```

`stat` – Display file status

```
charmgil@vm-VirtualBox:~/private/scratch$ stat hello.c
  File: hello.c
  Size: 90              Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d      Inode: 552269      Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/charmgil)   Gid: ( 1000/charmgil)
Access: 2019-03-18 21:28:08.105502374 +0900
Modify: 2019-03-05 04:02:54.566528235 +0900
Change: 2019-03-05 04:02:54.570530236 +0900
 Birth: -
```

- Identifier: filename and path
- Size, (physical) location, type
  - Size: #bytes, #blocks
  - Location: Inode (holds the mappings to device blocks)
  - Type: {regular, directory, symbolic link, pipe, socket, block/character device}

`stat` – Display file status

```
charmgil@vm-VirtualBox:~/private/scratch$ stat hello.c
  File: hello.c
  Size: 90              Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d      Inode: 552269      Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/charmgil)   Gid: ( 1000/charmgil)
Access: 2019-03-18 21:28:08.105502374 +0900
Modify: 2019-03-05 04:02:54.566528235 +0900
Change: 2019-03-05 04:02:54.570530236 +0900
 Birth: -
```

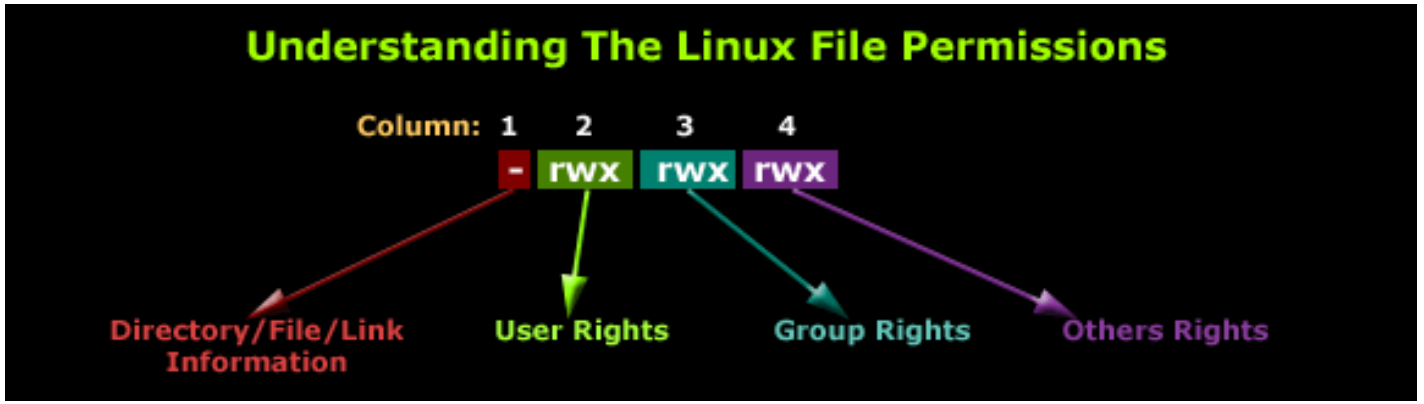
- Access control
  - Permissions (**r**ead/**w**rite/e**x**ecute)
  - Owner Uid (user) and Gid (group)
- Time
  - Last access time
  - Last modification (content update) time
  - Last change (content or metadata update) time

Permissions (Access Control)

- `$ ls -l`

```
charmgil@vm-VirtualBox:~/private/scratch/lab3$ ls -l
total 488
-rw-r--r-- 1 charmgil charmgil  74556 Mar 18 21:52 CRICKET.WAV
-rw-r--r-- 1 charmgil charmgil  31730 Mar 18 21:50 hgu_logo.jpg
-rw-r--r-- 1 charmgil charmgil   1440 Mar 10 17:16 kernel5.txt
-rw-r--r-- 1 charmgil charmgil 217993 Mar 18 23:08 lab3data.zip
-rw-r--r-- 1 charmgil charmgil  47207 Mar 18 21:50 linux_PNG1.png
-rwxr-xr-x 1 charmgil charmgil 114496 May 14  2018 x64-intro.pdf
```

Permissions (Access Control)



- Column 1: Type identifier (directory, file, link, …)
- Column 2: Permissions (r/w/x) for the owner (Uid)
- Column 3: Permissions for the group (Gid)
- Column 4: Permissions for everyone else

* Image obtained from: http://tecdistro.com/file-permissions-in-linux/

How to Modify Permissions?

- `chmod` – Change file modes or access control lists
  - Use-case #1: chmod <target>{+,-,=}<permission> <file name>
    - Target: u (user), g (group), o (others), a (all)
    - {+, -, =}: + (add), - (remove), = (set)
    - Permission: A combination of r, w, and x
    - Examples

Notation	Meaning
`u+x`	Add execute permission for the owner.
`u-x`	Remove execute permission from the owner.
`+x`	Add execute permission for the owner, group, and world. This is equivalent to `a+x`.
`o-rw`	Remove the read and write permissions from anyone besides the owner and group owner.
`go=rw`	Set the group owner and anyone besides the owner to have read and write permission. If either the group owner or the world previously had execute permission, it is removed.
`u+x,go=rx`	Add execute permission for the owner and set the permissions for the group and others to read and execute. Multiple specifications may be separated by commas.

How to Modify Permissions?

- `chmod` – Change file modes or access control lists
  - Use-case #2: chmod <octal digits> <file name>
    - Octal digits: Treat the permission mnemonics as a binary number

- - -	Not accessible	000	0
- - x	Executable	001	1
- w -	Writable	010	2
- w x	Executable & Writable	011	3
r - -	Readable	100	4
r - x	Readable & Executable	101	5
r w -	Readable & Writable	110	6
r w x	Fully accessible	111	7

- Examples: 751 = rwx r-x --x
  644 = rw- r-- r--

Agenda

- Coding in collaboration
- Version control systems
- Git
- GitHub

Agenda

- Coding in collaboration
- **Version control systems**
- Git
- GitHub

Version Control Systems

- Systems that help the management of changes to computer programs, documents, and other collection of information
  - Systems that record changes to a file or set of files over time, so that one can recall specific versions later
    - A.k.a. revision control or source control
  - Provide shared repositories for projects
    - Support and promote code sharing and collaboration
    - Allow to maintain multiple versions/branches of projects
  - Help simultaneous work on files
    - Offer easy ways to merge files with conflicts
  - Keep track of development projects; provide ways to “undo”

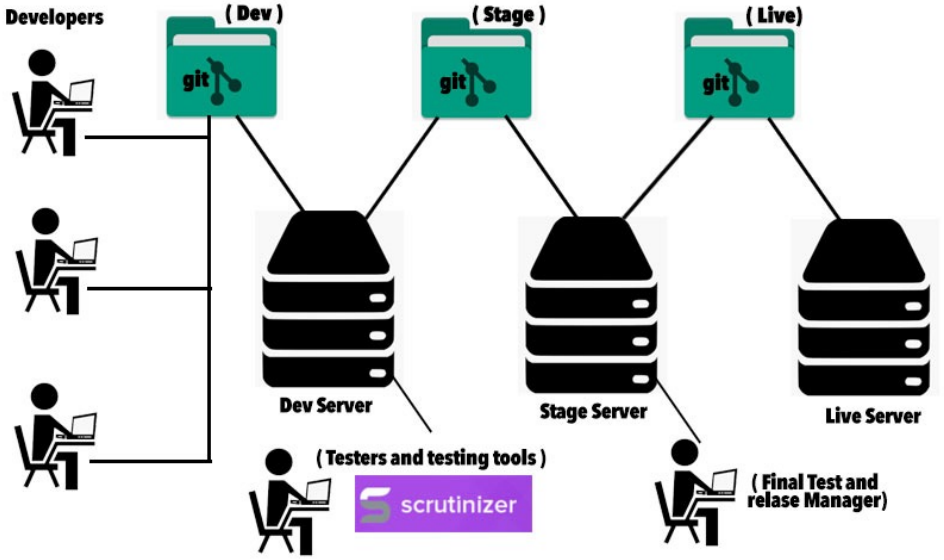
So, Version Controlled…

- It is not wrong to manually control versions using
  - Clever filenames: .bak, .orig, .old, .older, .oldest, .old.old, …
  - Version numbers: v1, v1a, v1aa, …
  - Date: 20200401_1223.docx, 20200403_0155.docx, …

* Image src: https://www.slideshare.net/visual28/version-control-58434966

Benefits of Version Control

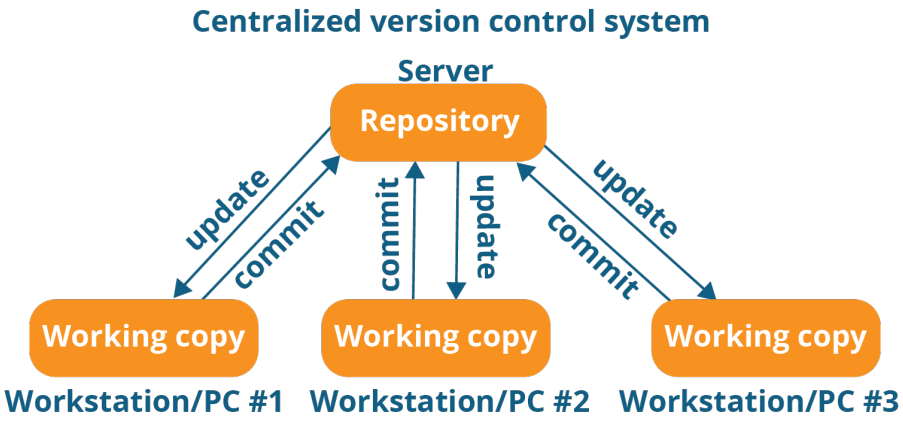
- Backup and restore
- Sync with multiple computers (multiple developers)
- Working in a group
- Safely create and test new features
- Ownership, credits, blame

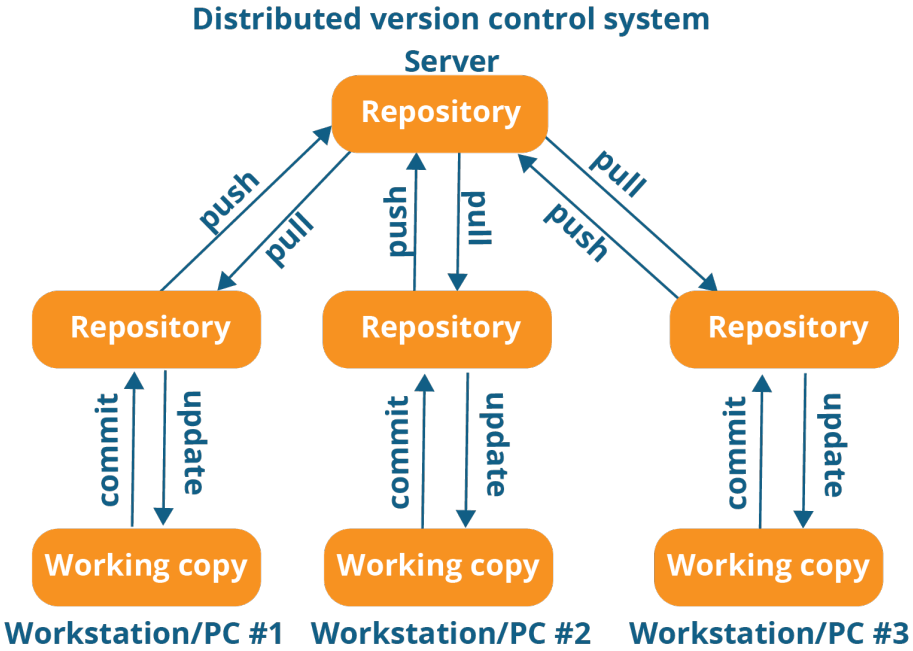


* Image src: https://www.helplazysam.com/tech/database-version-control-dbv/

Version Control Systems

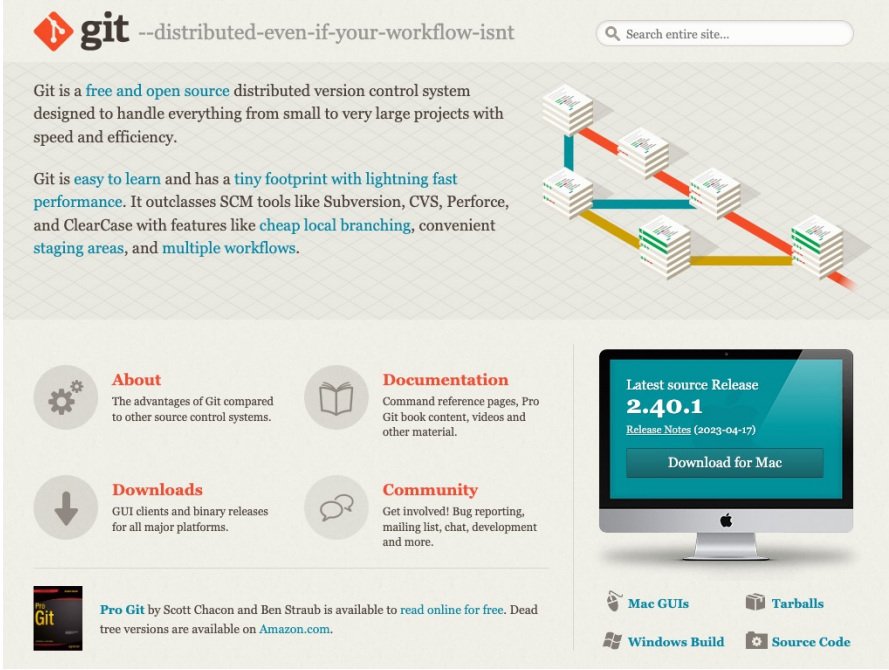
- Centralized VCS (cvs, svn) – a single server (repository)
- Distributed (decentralized) VCS (git)
  - No main server
  - All of the full history of the project is available once you cloned the project





git

- The mostly widely used, decentralized VCS
  - The Linux kernel project is running on Git
  - Available at: https://git-scm.com/



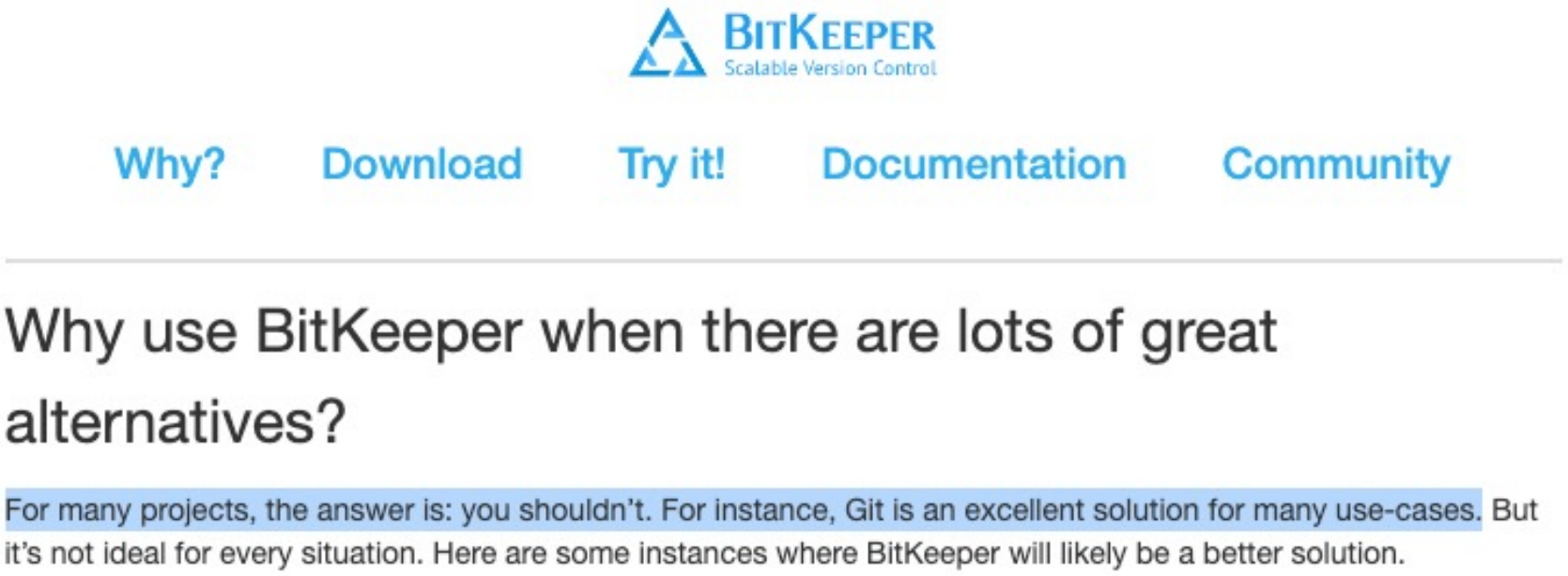
git

- Open-source project originally developed in 2005 by Linus Torvalds
  - After BitKeeper broke
    - The Linux Kernel project was hosted by BitKeeper
  - *A "git" is a cranky old man; Linus meant himself*
- A command line utility
- Sits on top of your file system and manipulates files
- A distributed version control system (DVCS)

BitKeeper
Scalable Version Control

* Image src: BitKeeper; https://insights.dice.com/2018/09/18/linus-torvalds-apology-highlights-soft-skills-necessary/
https://twitter.com/talitapagani/status/511353454443626496

C.f., BitKeeper



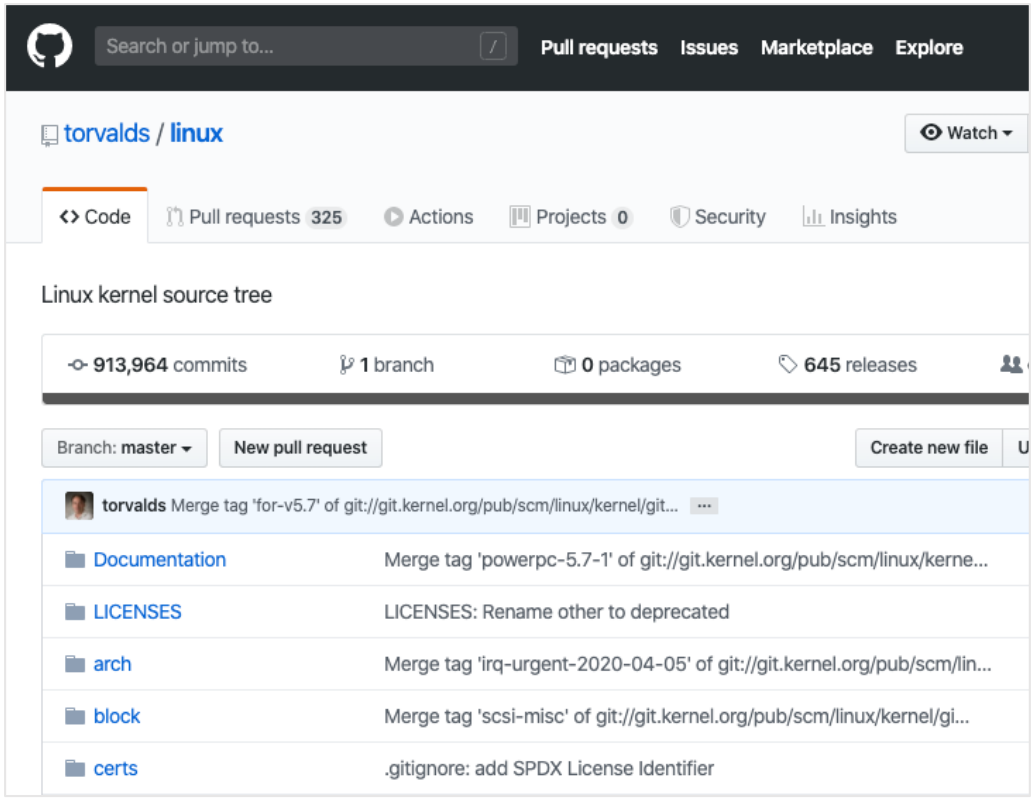
git

- Git is
  - A distributed version control system (DVCS)
  - Free and open source
  - Designed to handle everything from small to very large projects
  - Easy to learn and maintain

  - Git takes a snapshot of your repository at a given point in time
    - This snalshop is called a commit
  - Git is optimized for local operation
    - Clone a repository = a copy of the entire repository + its history
  - Branches are lightweight and cheap
  - Git is explicit: No auto-saves or auto-syncing with the remote

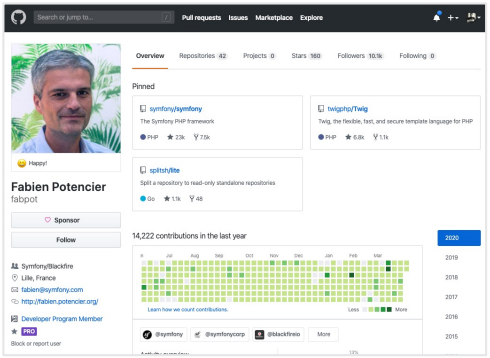
GitHub

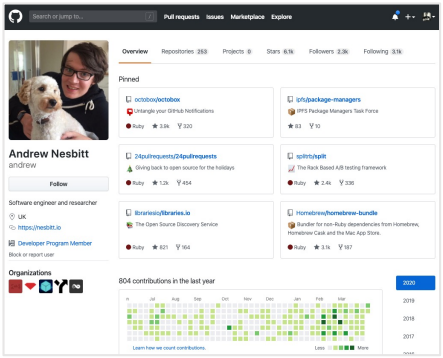
- An online hosting service (of Microsoft) for software development and version control using Git; provides:
  - A collaboration platform (Distributed version control of Git)
  - Multi-user access control
  - Bug tracking
  - Software feature requests
  - Task management
  - Continuous integration
  - Wikis and bulletin board

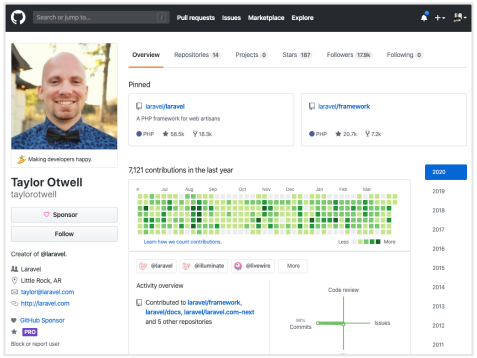


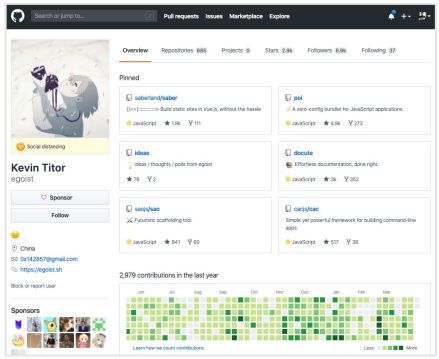
GitHub

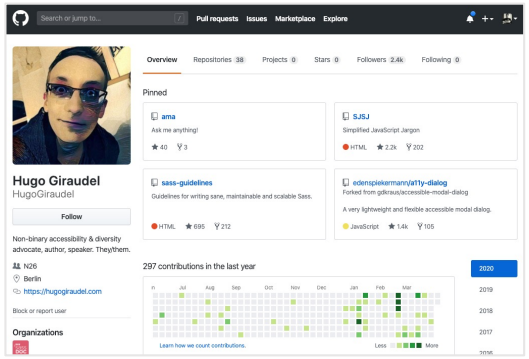
- GitHub.com = Git + public repositories + social network
  - Something you want to tell on your resume

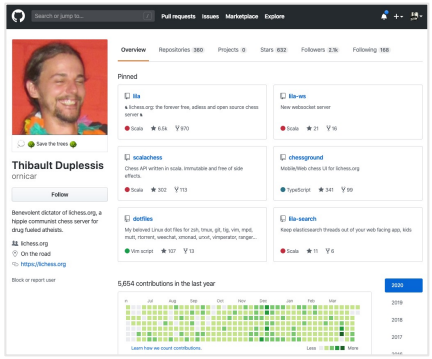








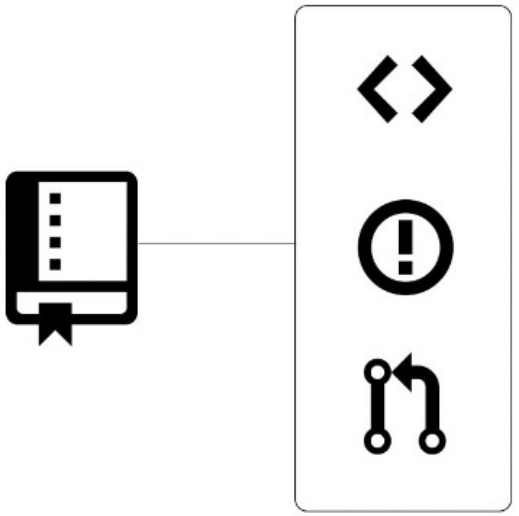




* Image src: https://github.com/fabpot; https://github.com/andrew; https://github.com/taylorotwell; https://github.com/egoist; https://github.com/HugoGiraudel; https://github.com/ornicar

A GitHub Repository

- Repository: the msot basic element of GitHub
  - Like a project's folder
  - Key components
    - Code
    - Issues
    - Pull requests
    - Wiki
    - Pulse
    - Graphs
    - README.md



Working Locally with Git

Get Your Initial Make Project

- One header file (.h) – mylib.h
- Two source files (.c) – main.c, mylib.c

main.c

```c
#include <stdio.h>
#include "mylib.h"

int main(void){

  int a = 3, b = 5;
  printf("(initial) a=%d, b=%d\n", a, b);

  swap(&a, &b);
  printf("(swapped) a=%d, b=%d\n", a, b);

  return 0;
}
```

mylib.h

```c
#ifndef  _MYLIB_H_
#define  _MYLIB_H_

void swap(int*, int*);

#endif
```

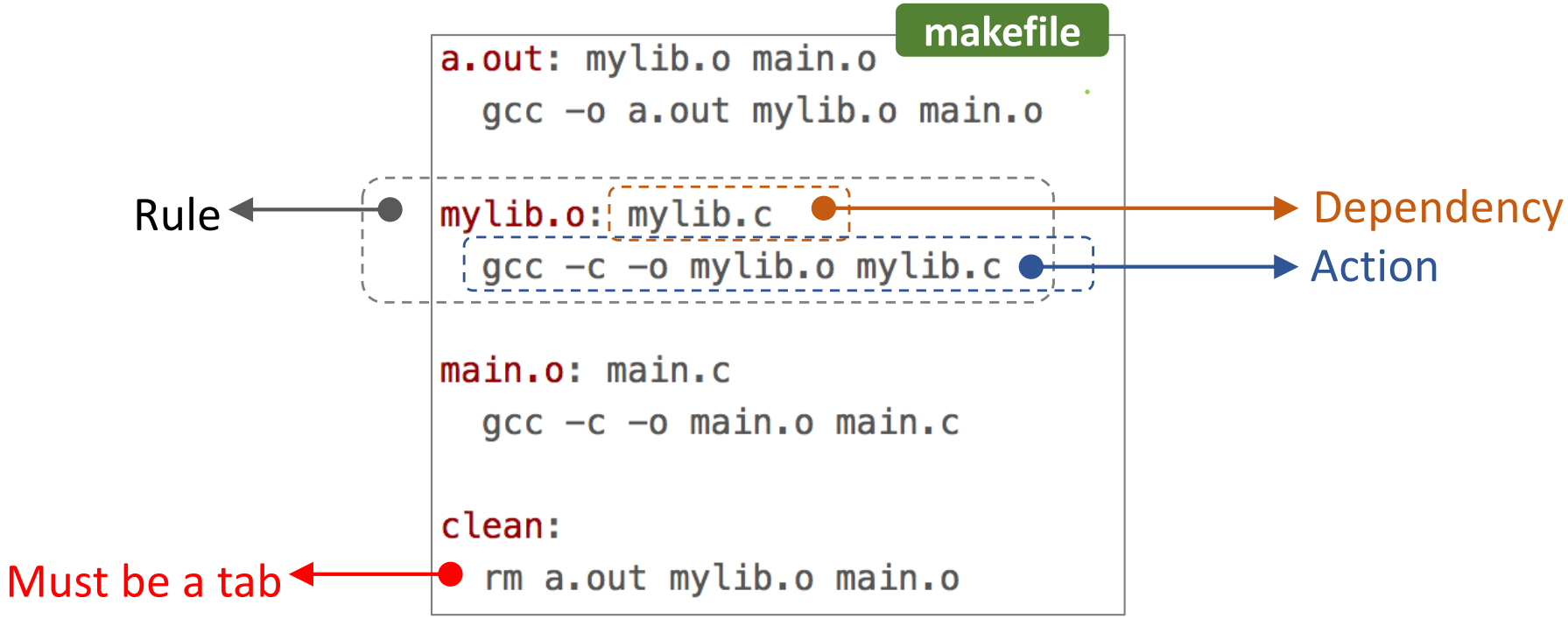
mylib.c

```c
#include "mylib.h"

void swap(int* a, int* b){
  int tmp = *a;
  *a = *b;
  *b = tmp;
}
```

Get Your Initial Make Project

- An example *makefile*



Check If You Have Git

- Make sure you have a Git client on your machine

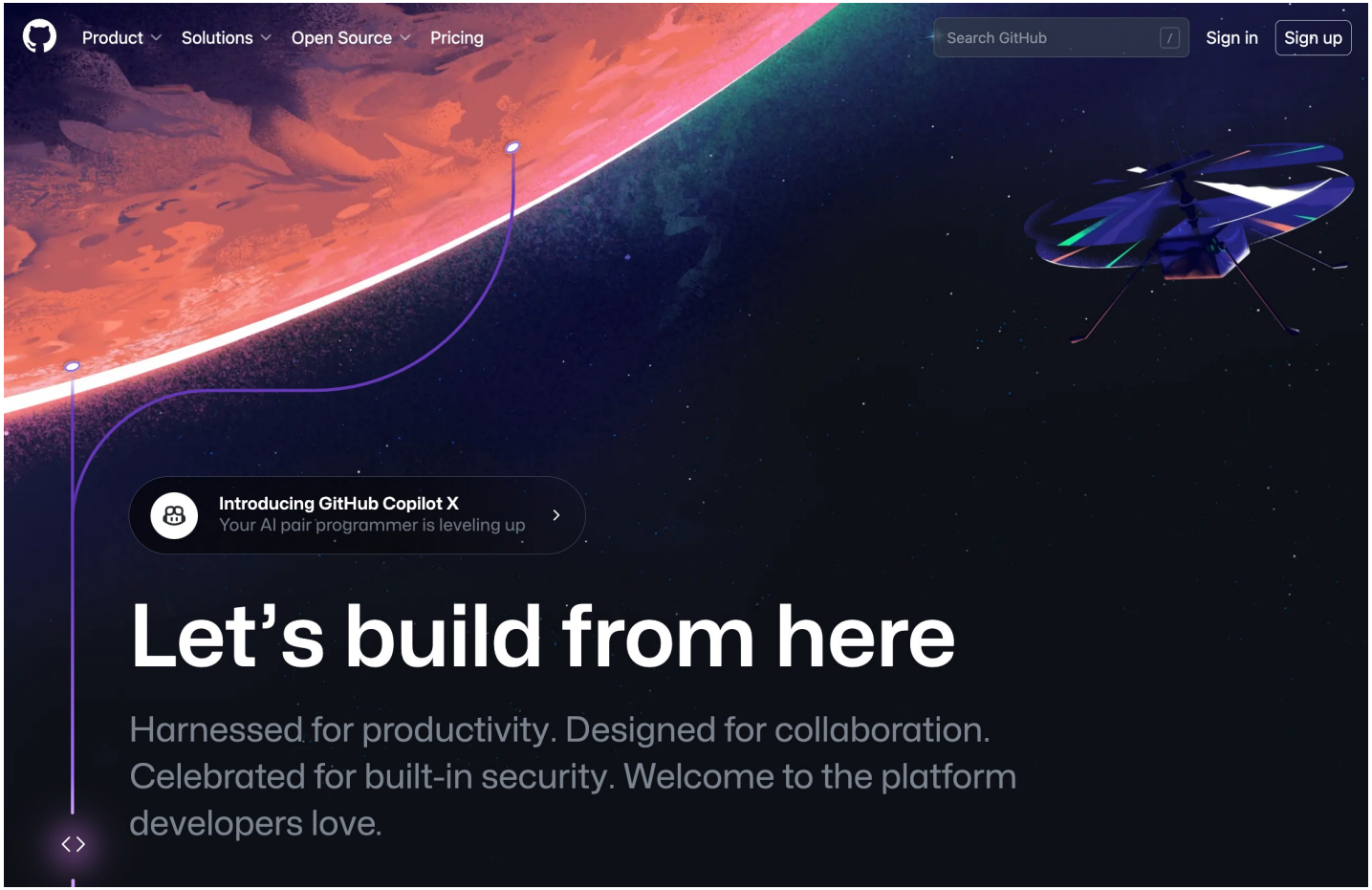
```
$ git --version
git version 2.7.4
```

- Add your identity

```
$ git config --global user.name "YOUR NAME"
$ git config --global user.email ACCOUNT@DOMAIN.COM
```

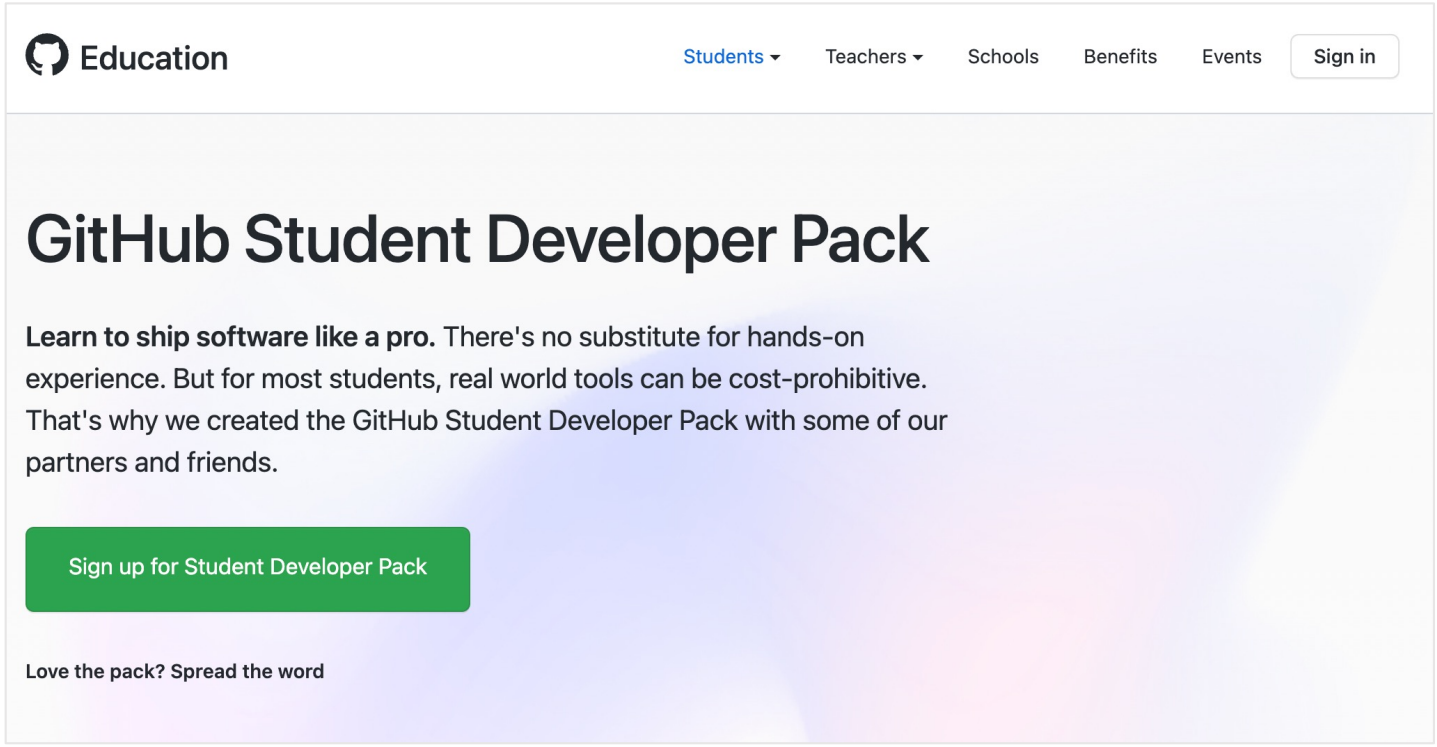
Step 0: First Things First

- Create an account on GitHub.com



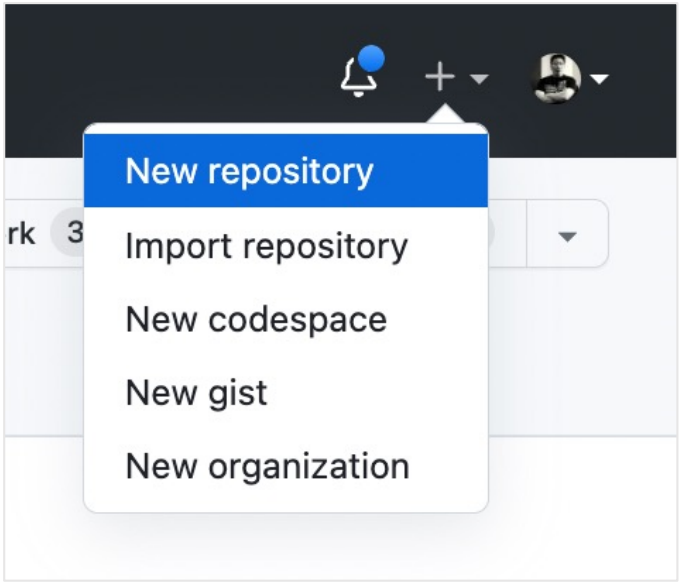
Step 0: First Things First

- Don’t forget to get your bonus stuff
  - GitHub.com offers a free software and service package for students
  - https://education.github.com/pack



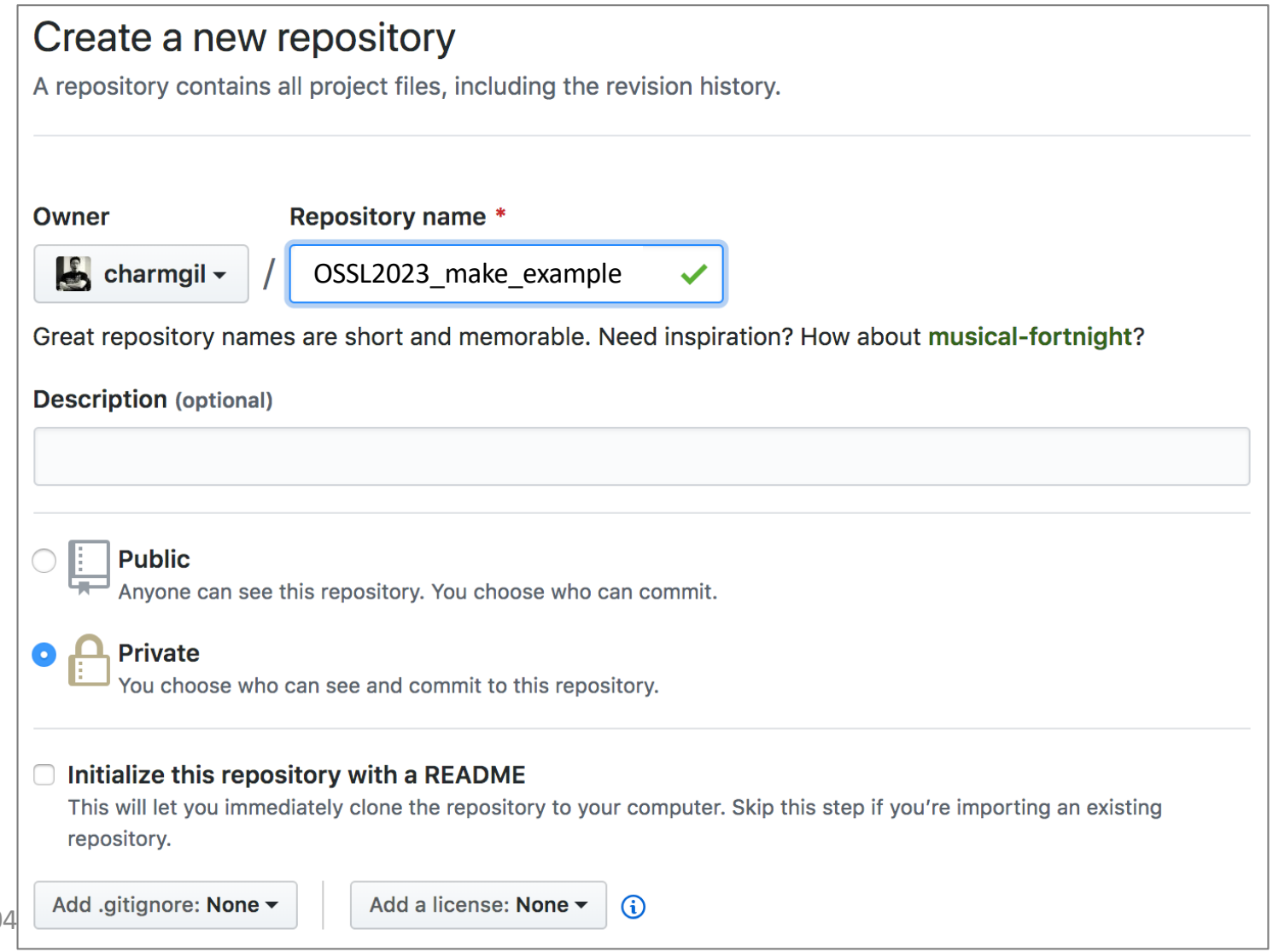
Step 1: Create a New Repository

- Click on the + button (upper-right corner) and select "New repository" (repo)



Step 1: Create a New Repository

- Create a new repository (repo) on GitHub.com



Step 1: Create a New Repository

- Create a new repository (repo) on GitHub.com

Step 2: Initialize Your Project Directory

- 'git init' under your project directory

```
$ ls
main.c    makefile    mylib.c    mylib.h
$ git init
Initialized empty Git repository in
/Users/charmgil/Projects/CODE.c/github_example/.git/
```

- Directory .git is created – This is your local repo

```
$ ls -a
.       .git       makefile  mylib.h
..      main.c     mylib.c
```

Step 2: Initialize Your Project Directory

- In case you want to stop tracking your project, simply remove the .git directory

```
$ ls -a
.       .git       makefile  mylib.h
..      main.c     mylib.c
$ rm -rf .git
```

Step 3: Check the Status

- 'git status' – Check the status of repo

```
$ git status
On branch master

No commits yet

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

    main.c
    makefile
    mylib.c
    mylib.h

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

Step 3: Check the Status

- 'git status' – Check the status of repo

```
$ git status
On branch master

No commits yet

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

    main.c
    makefile
    mylib.c
    mylib.h

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

No commits yet; as we have just initialized it

Step 4: Create .gitignore

- In .gitignore, list items that you don't want to track
  - For example, exclude *.o files, which are the intermediate object files after compilation (*your collaborators do not need *.o files*)

```
$ touch .gitignore        # create a file
$ vi .gitignore           # list files/dirs to exclude

$ cat .gitignore
*.o
```

Step 5: Add Local Files to git

- Add the project files, that you want to track, to Git
  - `git add -A` or
    `git add .`

```
$ git add -A
```

Step 5: Add Local Files to git

- Commit the changes to your local git repository (repo)
  - `git commit -m “first commit”`

```
$ git commit -m "first commit"
[master (root-commit) ef5376a] first commit
 5 files changed, 38 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 main.c
 create mode 100644 makefile
 create mode 100644 mylib.c
 create mode 100644 mylib.h
```

Step 6: Associate the Repositories

- We have created two repos; We want to associate them
  - One on the local machine
  - Another on GitHub.com

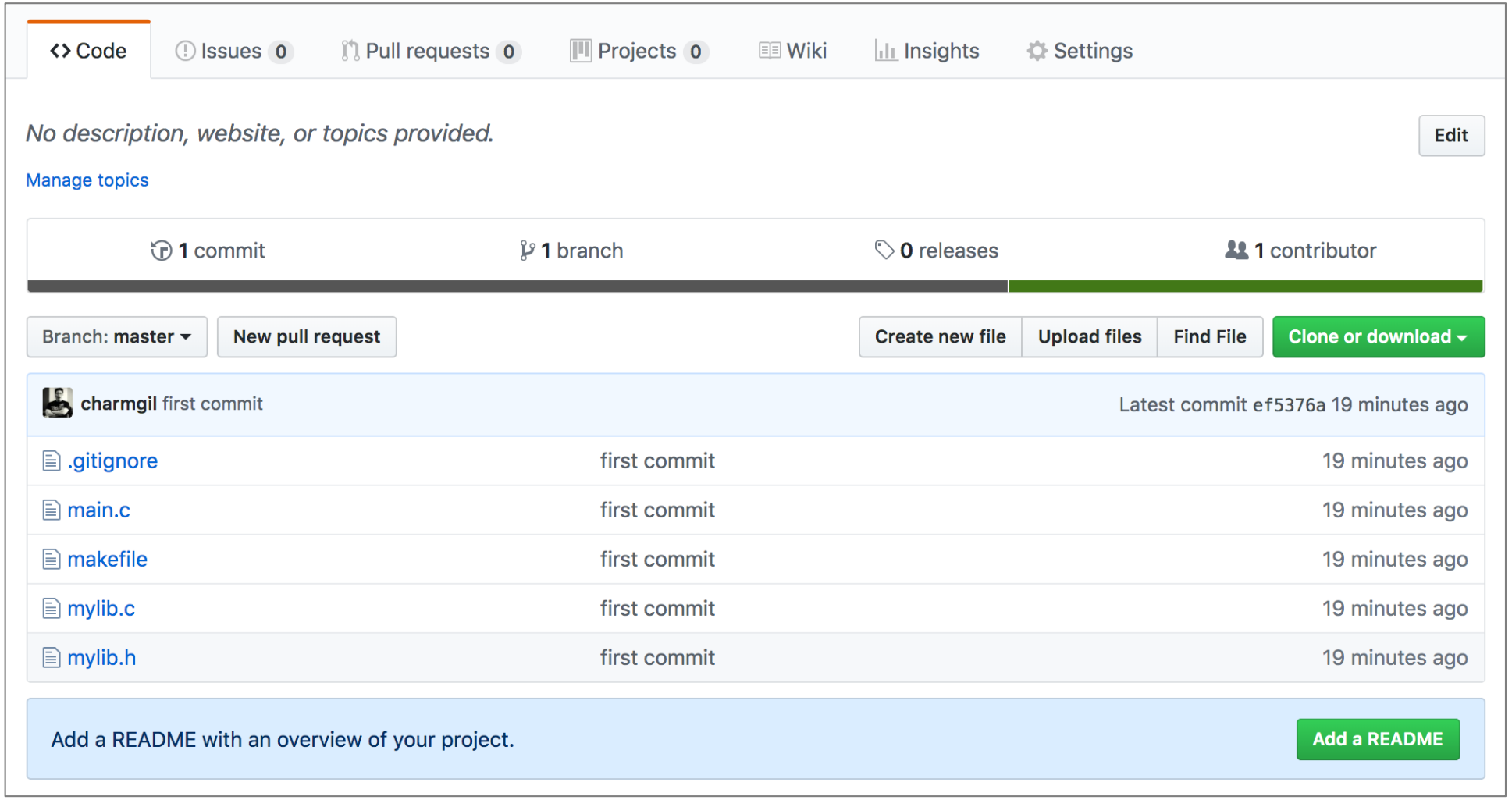
```
$ git remote add origin https://github.com/charmgil/
OSSL2020_make_example.git
```

Step 6: Associate the Repositories

- We have created two repos; We want to associate them
  - One on the local machine
  - Another on GitHub.com

```
$ git push -u origin master
Counting objects: 7, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (7/7), 701 bytes | 701.00 KiB/s,
done.
Total 7 (delta 0), reused 0 (delta 0)
To https://github.com/charmgil/OSS2020_make_example.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master'
from 'origin'.
```

Completed: Your Project is on GitHub

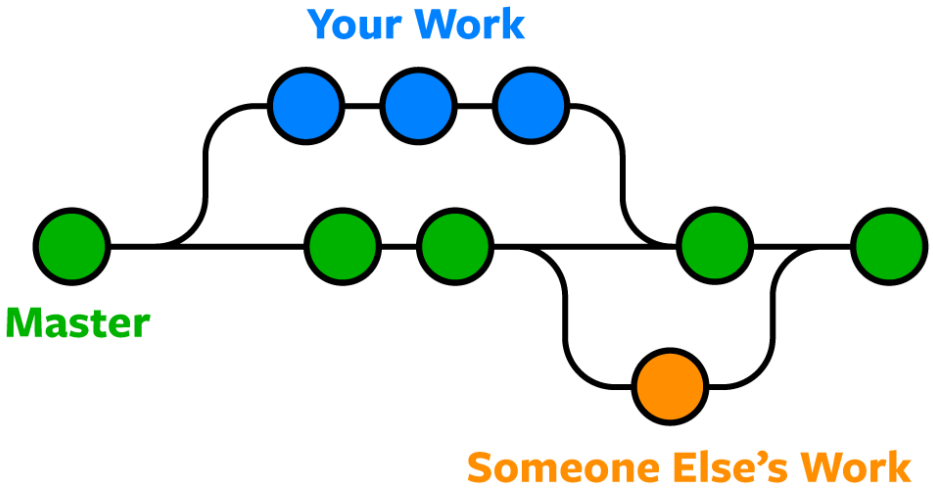


Your Project is on GitHub

- You have established your remote VCS on GitHub
  - You can access your project from anywhere in the world
- Now every developer in the world can watch your development!

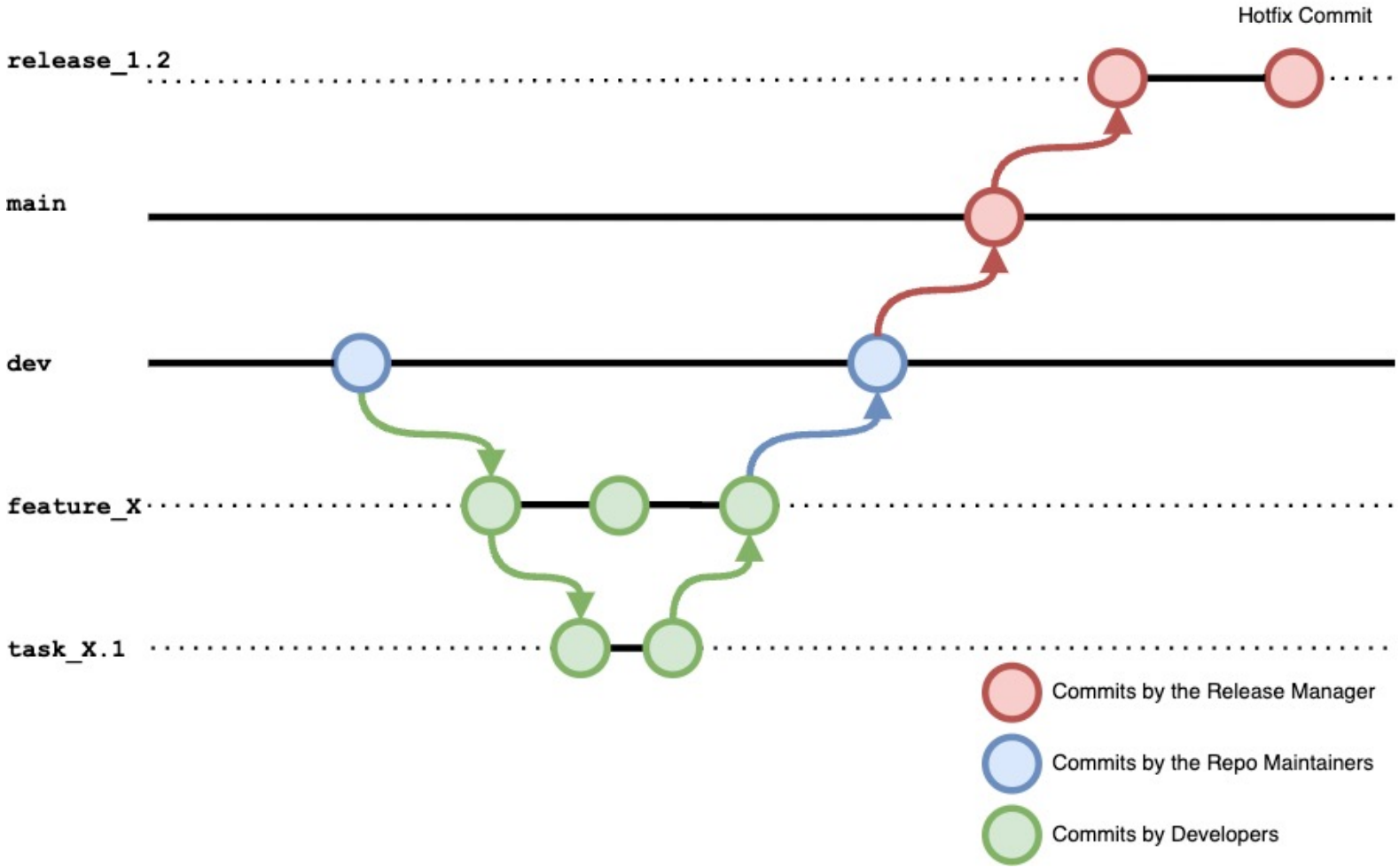
Other Important Functions

- Branching and merging



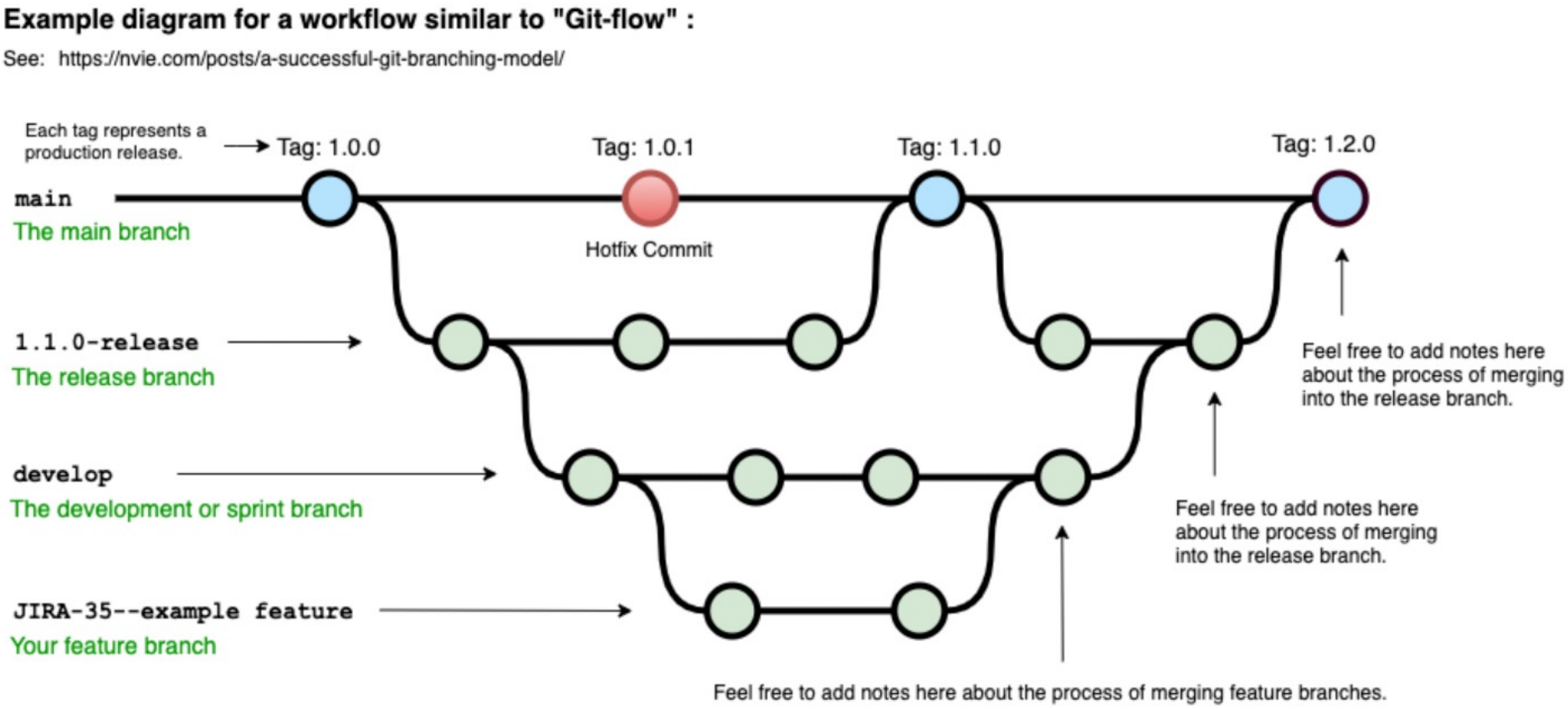
Other Important Functions

- Branching and merging



Other Important Functions

- Branching and merging



Other Important Functions

- Fork: Full copy of a repository residing on a different account