Lec 2: Background

IS521: Information Security Laboratory

Sang Kil Cha
Version Control System
Git?

• Created by Linus Torvalds\textsuperscript{1} in 2005.
• \textit{Distributed} version control system.

\textsuperscript{1}The creator of the Linux kernel.
Version Control System (VCS)

A system that records changes to a set of files over time.
Version Control System (VCS)

A system that records changes to a set of files over time.

Diagram:

```
Version 1
    ▲
Version 2
```
Version Control System (VCS)

A system that records changes to a set of files over time.
The Simplest VCS

- Create a ZIP file for each version.
- Keep all the ZIP files on a storage or a database.
The Simplest VCS

- Create a ZIP file for each version.
- Keep all the ZIP files on a storage or a database.

What's wrong?

- Store only diffs.
- Create a *local* DB for each individual file to record diffs.
- Cannot share changes with a team or other developers.
An Even Better Approach: CVS (1985)

- CVS stands for Concurrent Version System.
- Keep track of changes in a set of files.
- Use *client-server* model.
- Programmers can remotely share their work!
CVS is Still Problematic

- Commits are **not atomic**.
  - Interrupted commits can corrupt the repository.
- CVS only tracks modification on a **file basis**.
  - It is difficult to follow the history of a project. Why?

- **SubversioN** (SVN).
- Maintain diffs for a set of files.
- Support atomic commits.
- Client-server model (i.e., centralized VCS) as CVS.

- **SubVersion** (SVN).
- Maintain diffs for a set of files.
- Support atomic commits.
- Client-server model (i.e., centralized VCS) as CVS.

Centralized (SVN) vs. Distributed (Git)?
Centralized VCS

[Diagram showing version control system with server DB and clients getting versions from server]
Distributed VCS

Server DB

Version 1
Version 2
Version 3

Get copy

Client A

Version 1
Version 2

Get copy

Client B

Version 1
Version 2
Version 3
Why Distributed VCS?

• Can work offline.
  - Don’t need to create a single large diff.
  - We can locally manage *many, but small* versions.

• Fast.
  - Most actions are done locally.

• Easy to share across small groups.
  - Small group of developers can share changes before showing them to everyone.
Any Disadvantages of Distributed VCS?

• Slow initialization.
Any Disadvantages of Distributed VCS?

- Slow initialization.
- Requiring larger disk space than centralized VCS.
Any Disadvantages of Distributed VCS?

• Slow initialization.
• Requiring larger disk space than centralized VCS.

But, overall, more advantageous!
Git (2005) = Distributed VCS

Simple design principles:

• Be **simple** and **fast**.

• Support **fully distributed**, non-linear development.

• Must be able to **handle large projects** like the Linux kernel.
Using Git
Initialize Local Repository

Create a new local repository from scratch.

> git init

Copy a remote repository to the current directory (i.e., local repository).

> git clone <URL>
Initial Configuration

- `/etc/gitconfig`: per-system configuration.
- `~/.gitconfig`: per-user configuration.
- `.git/config`: per-repository configuration.

Per-system Setup (N.B. use your name and email).

```bash
> git config --global user.name "Sang Kil Cha"
> git config --global user.email "sangkilc@kaist.ac.kr"
```
You want to use your own editor to write a commit message.

Example: Vim On Linux.

```sh
> git config --global core.editor vim
```

You can also a GUI editor on Windows. For example, see: https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/How-to-set-Notepad-as-the-default-Git-editor-for-commits-instead-of-Vim.
Four States of a File in Git

1. **Untracked** state: the file is not under version control.
2. **Committed** state: the file is stored in a local repository.
3. **Modified** state: the file is modified, but not committed yet.
4. **Staged** state: the file is modified, and marked to be committed in the next snapshot.

Check the current state of files.

```bash
> git status
```
git add <file>

- **Untracked** → **Staged**.
- **Modified** → **Staged**².

²*Staged* files immediately become *Modified* whenever we modify the file with an editor.
git commit

• **Staged → Committed.**

• We need to write a commit message with our editor\(^3\).

• We can also add a commit message directly from a command line:
  git add -m "Commit message".

\(^3\)In case we exit the editor without saving the commit message to an automatically generated file, Git will abort the commit, and the files should remain **Staged.**
git log

Check the **Committed** versions.

commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date: Mon Mar 17 21:52:11 2008 -0700

    changed the version number

...
Check the **Committed** versions.

commit ca82a6dff817ec66f44342007202690a93763949  
Author: Scott Chacon <schacon@gee-mail.com>  
Date: Mon Mar 17 21:52:11 2008 -0700

changed the version number

...
Sync with Remote

The *Committed* versions can be synced with the remote repository.

**Push the local commits to remote.**

```
> git push origin master
```

**Pull new commits from the remote.**

```
> git pull origin master
```
git push origin master

Remote name  Branch name
Branching

Git branch is a pointer to one of the commits.
Branching

Git branch is a pointer to one of the commits.

The default branch name is `master`, which always points to the last commit.
We can make a new branch `mybranch` from `master`:

```bash
git branch mybranch
```
We can make a new branch `mybranch` from `master`:

```
  git branch mybranch
```
There is another default pointer, called HEAD.

- **HEAD** points to the current branch.
- We can move **HEAD** around by checking out a branch.
  - `git checkout <branch name>`
Moving the HEAD to mybranch.

> git checkout mybranch
HEAD (cont’d)

Moving the HEAD to mybranch.

> git checkout mybranch
Additional Branching Commands

Create a branch and move HEAD to the branch.

> git checkout -b mybranch

Remove a branch.

> git branch -d <branch name>
Branching Exercise

Start from an empty git repository, can you make it look as above?
Remote

• A **remote** means a remote Git repository.
• When we clone a repository, the **origin** remote is automatically created, which is a default remote.
• A **remote** has its own branch.
  - “origin master” means the master branch in the **origin** remote.
• There is no fundamental difference between a local branch and a remote branch, except that **HEAD** cannot point to a remote branch.
Merge Two Development Histories

> git checkout master  # HEAD should point to master.
> git merge mybranch
Merge Two Development Histories (cont’d)
Pulling Internals

The “git pull” command is a shorthand for “git fetch” followed by “git merge”. In other words, it first fetches from a remote branch, and try to merge changes in the branch into the current branch.
GitHub
GitHub?

GitHub hosts Git repositories. It also provides free web-based services.
1. Read and complete the exercise:
   https://guides.github.com/activities/hello-world/

2. Read:
   https://guides.github.com/introduction/flow/
You will use Git and GitHub for developing your CTF problems. Each student is given a private repository at first, but later, we will publicize your source code just before we start a CTF event (not your git logs though).
Docker-based CTF
Docker Container

Isolated environment for running applications.
Container vs. VM

**Container**
- App A
- Lib A
- Docker
- OS
- HW

**VM**
- App A
- App B
- App C
- Lib A
- Lib B
- Lib C
- Guest OS
- OS
- HW
Basic Terminologies

A **docker image** is a virtual matter, which defines an execution environment. It can be instantiated by `docker run` to create a **docker container**. We can create multiple containers from an image.
Basic Commands

Listing Docker Images

```
docker images
```

Listing Docker Containers

```
docker ps
```
Example

```csharp
open System

[<EntryPoint>]
let main _argv =
    Console.WriteLine "1 + 1 = ?"
    Console.ReadLine () |
    |> Convert.ToInt32 |
    |> fun n ->
        if n = 2 then IO.File.ReadAllText ("/var/ctf/flag") |
        |> Console.WriteLine
        else Console.WriteLine ()
    0
```
FROM mcr.microsoft.com/dotnet/core/sdk:3.1

RUN apt-get update && apt-get install -y xinetd
RUN mkdir -p /var/ctf /app
COPY flag /var/ctf/
COPY myservice /etc/xinetd.d/
COPY Program.fs /app
COPY *.fsproj /app

RUN echo "myservice 4000/tcp" >> /etc/services
RUN service xinetd restart

WORKDIR /app
CMD ["xinetd", ":-dontfork"]
Service Configuration File

```plaintext
service myservice
{
    flags = REUSE
    socket_type = stream
    wait = no
    user = root
    server = /usr/bin/dotnet
    server_args = run --project /app
    disable = no
    port = 4000
}
```
How to Instantiate a Container?

```
docker run --rm <image id> -p 4000:4000
```

Redirect the port from 4000 to 4000 using the \(-p\) option.
Conclusion
• Git is a powerful version control system.
• You will use it to develop your CTF problems.
• Each CTF problem should run a network-hosted application, which takes in a network input and returns a flag when exploited.