Tools

1. In your opinion, what's helpful about version control systems? What's annoying about them?

Here is a quick summary of my view on the high-lights and low-lights of a version control system in general.

Benefits:

- **Handy versioned changes:** Keep the actual historical changes and change history and provide the ability to restore the code to a certain snapshot. This enables the developer to experiment at will, then converge to the right solution as long as she regularly checks in the code at the appropriate time during the development lifecycle;
- **Create dev branches easily:** The developers can opt to work in their own branches if they want to experiment with changes without having to worry about interfering with others;
- **Relative ease to merge:** The code changes from selected or all other developers, facilitating the right amount collaboration without having to force them to coordinate constantly. It also eases up the merges between different branches, for most cases;
- **Allow for release versioning and branching:** It enables the engineering team to build reproduceable solutions by using tags or labels as well as suitable branching strategies while other development is still on-going;
- **Facilitate parallel releases:** It supports multiple versions at the same time of software releases if the circumstance requires an organization to do so without too much cost from the source code configuration management perspective;
- **Provide ability to hotfix past releases:** It helps the organization to fix production problems of the previously released solutions (through well-maintained release branches) without having to worry about the code changes introduced after its original release;
- **Track changes at a low cost:** Changes are tracked and restored very efficiently as only the incremental deltas are persisted and a mechanism by the system to reconstruct the entire file(s) at ease;
- **Enable vastly distributed development:** In the case of a vastly geographically distributed organization, it is very convenient for teams and individuals to collaborate and share work.

Downsides:

- **Manual merge conflicts:** When a lot of changes made by many people to the same segments of the same files due to lack of coordination beforehand, it can be a daunting task for someone to make a coherent merge;
- **Too easy to branch:** Developers may be too liberal in producing new branches, which could be confusing to themselves or their teammates or other teams;
- **Challenges in managing and synchronizing branches:** Maintain a manageable set of branches and branching strategy can be challenging. At times it may become almost impossible to reconcile the conflicting changes made across a myriad of poorly improvised branches;
- **Difficulty in propagating common changes:** Critical function or security updates that are applicable to a long series of branches propagation can be a daunting effort;
- **Compatibility issues among different tools.** Different version control systems have their own way of doing things and may not be compatible with one another. Thus, migrating from one version control solution to another is not an easy task if an organization decides to change the tools to be used down the road.

2. *What are some pros and cons of using Docker to develop, test, and deploy software?*

Docker is a great technology and it enables a lot of different approaches to address the modern scalability and isolation problems as the cloud applications become more complicated. However, it does have its own plus and downsides.

Pros:

- Light-weight, dedicated run-time environment for the particular or related applications, servers or datastores (To be collectively referred to as applications);
- Encapsulation and protection from interferences between or among applications where conflicts might exist between the underlining libraries due to peculiarities or differences in the dependencies upon the libraries among the applications;
- Consistent and light-weight environment across the teams for better reproducibility and consistency. This definitely improves the team productivity;
- Easy and dedicated data volumes may help reduce the reliance on mocks for many automated unit, semi-unit, integration or BDD testing;
- Better support for microservice architectures, which helps horizontal scaling and divide-and-conquer for both development and testing;
- Widespread community support and knowledge base for the team to leverage on and reuse.

Cons:

- Replicated installations of shared (especially large) libraries or binaries for the same or similar applications residing in the same or similar, but separate container instances;
- Complicated inter-container communications, security postures and exposures. The container themselves are not very fortified as compared to the traditional virtual machines where the OS vendors have well established processes to harden the system and provide security updates. Oftentimes, individually or small-team created containers with limited expertise in securing the entire container. This elevates the importance of the security protection of container engine, and provides attackers a focused target;
- Too many containers to manage and monitor, especially in staging and production environments;
- Single point of failures for all containers running on top of the same Docker engine, which, in turn, runs in a host OS environment. If someone manages to intentionally or unintentionally bring down either the Docker engine or the host OS, the results would be catastrophic;
- Potential interferences in data sharing among different containers, which requires careful analysis, configuration and monitoring;
- Hard to debug the "black-box" operations inside a container when running in the detached mode;
- Inconsistent support by Docker engine in Mac OS and Windows host OSes as compared to Linux-flavored ones.

3. *How do you choose which language to use for a given task? How did you choose the language 1 for the programming exercise above?*

Programming language selection criteria:

When choosing a language for a given task, I normally consider the following aspects:

- How well the language supports the functionality and features needed for the project. For instance, if performance is the ultimate goal and/or space is a major factor, then compact and fast

compiled language might be the best choice. This is in great contrast to the scripting languages used for cloud development;
- What long-term prospects of getting continued support either through a vendor or the community;
- How the expertise of a language fares on the team and/or how easy it is to ramp up;
- How well it is suited to readily develop and adequately test the implementation, such as automated unit and BDD testing;
- Are there any benefits and/or downsides for supporting the efforts in CI/CD and DevOps in the case of cloud solutions;
- How the language still enjoys and it continues to enjoy enthusiasm from the industry. For instance, is there a reasonable repository of available libraries or packages for the team to leverage?
- Cost associated with developing and maintaining the solution to the project down the road;
- Current technology trend in the software industry, and how it supports the organization or company to attract the top talents;
- Internationalization and/or localization support if that is a key factor for the success of the business.

Reasons to choose Python for the programming task:

Obviously, it was specified as a requirement in the assignment beforehand. However, I would have chosen it myself as well had the language requirement not given. The main factors considered are:

- Python is quite mature and very good for file operations, text processing, list and other collections handling;
- It is easily to develop and test. One can implement a solution by developing a few lines of code in Python whereas another language like C++, Java or C# may take quite more effort to do so. JavaScript can be used as well, but its browser-oriented roots make it slightly harder to use for general purpose programming. Less code means smaller chance to introduce bugs, and less effort required to test and debug the implementation;
- I have expertise in Python to design, implement and test the code quite easily;
- Docker support for Python is good, and invoking Python scripts from the container shell is pretty convenient as well.