Version Control and GitHub

```
$ echo "Data Sciences Institute"
```

Prerequisites:

- Git version ≥2.39
- Git Credential Manager
- GitHub account

Key Texts:

- Chacon and Straub, 2014, Pro Git, 2nd Edition.
- Timbers, Campbell, Lee, 2021, Data Science: A First Introduction, https://ubc-dsci.github.io/introduction-to-datascience/

References

- Chacon and Straub: Chapter 1
- Timbers: Chapter 12.3 - 12.4, 13.3.1

`Version Control`

What is Version Control?

Version control is a system:

- that records changes to a file or a set of files over time
- that enables recall a specific version

We may already do this by copying files to another directory to save past versions. While it is simple, it lacks flexibility and complexity.

Why version control?

Version Control Systems (VCS) can do a number of things and can be applied on nearly any type of file on our computers:

- revert files to a previous state
- revert entire project to a previous state
- compare changes over time
- see who modified something last
- who introduced an issue and when
- recover lost files

Why specialized version control for software teams?

- Robust software is documented as it is written
    - Log changes and reasoning for why changes are made
- Working in teams requires code-specific version control
    - Changing one part of a code project can affect behaviour in seemingly unrelated features
    - In-progress state of one component can render the entire program temporarily unusable (e.g. syntax error)

Questions?

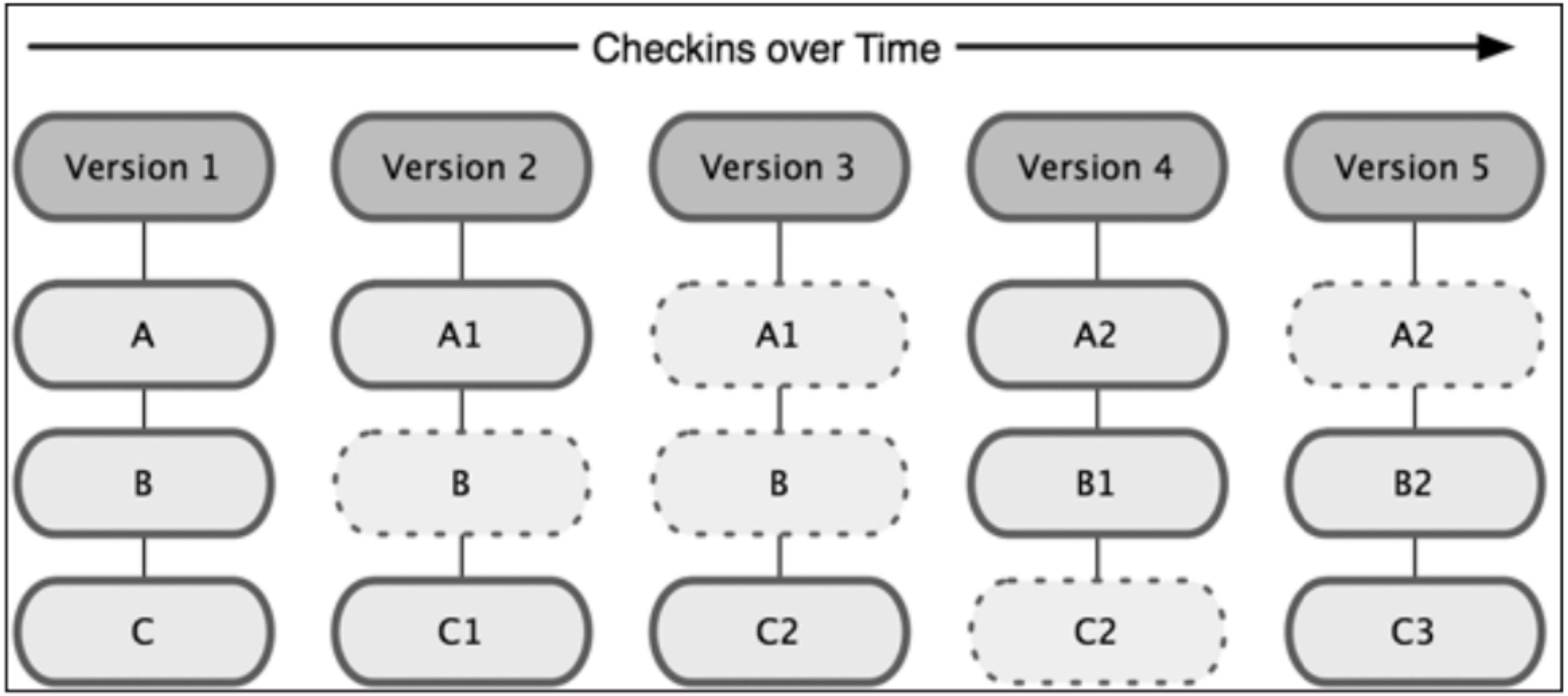
`Git`

Git Basics

Git is the most common VCS in modern coding teams.

Git is efficient with storage

- Git stores snapshots of files in your project directory
- If files have not changed, Git does not store the file again, it links to the previous identical file already stored.


Checkins over Time
Version 1
Version 2
Version 3
Version 4
Version 5
A
A1
A1
A2
A2
B
B
B
B1
B2
C
C1
C2
C2
C3

Git can operate locally

Most operations on Git only need local files and resources to operate. Git also keeps the entire history of our projects on our local disks meaning we can see changes made months ago without a remote server.

We also don't need to be connected to the server to get work done, rather we only need to be connected when we want to upload our work.

Git assures integrity

Git uses a check-summing mechanism called *SHA-1 hash* which is calculated based on the contents of a file or directory structure in Git. It looks somehting like this:

```
24b9da6552252987aa493b52f8696cd6d3b00393
```

This checksum means it's impossible to change the contents of any file or directory without Git knowing about it.

Git generally only adds data, making it fairly difficult to lose data once we've committed, which we'll learn about later.

The Three States

There are three main states that our files can reside in:

- Committed:
    - data is safely stored on local database
- Modified:
    - file has been changed but not yet committed
- Staged:
    - modified file has been marked to go into the next commit

The Three Main Sections

There are three main sections to a Git project:

- The Git directory
- The working directory
- The staging area

The Git Directory

The Git directory is where Git stores the metadata and object database for our projects. It is what is copied when we clone a repository from another computer.

The Working Directory

The working directory is a single checkout of one version of our projects. These files are pulled out of the compressed database in the Git directory and placed on the disk for us to modify.

The Staging Area

The staging area is a simple file that stores information about what will go into our next commit.

Workflow

A basic workflow will look something like this:

1. Modify files in our working directory
2. Stage the files in the staging area
3. Commit the changes which takes the files from the staging area and stores them on the Git directory.

Questions?