VERSION CONTROL

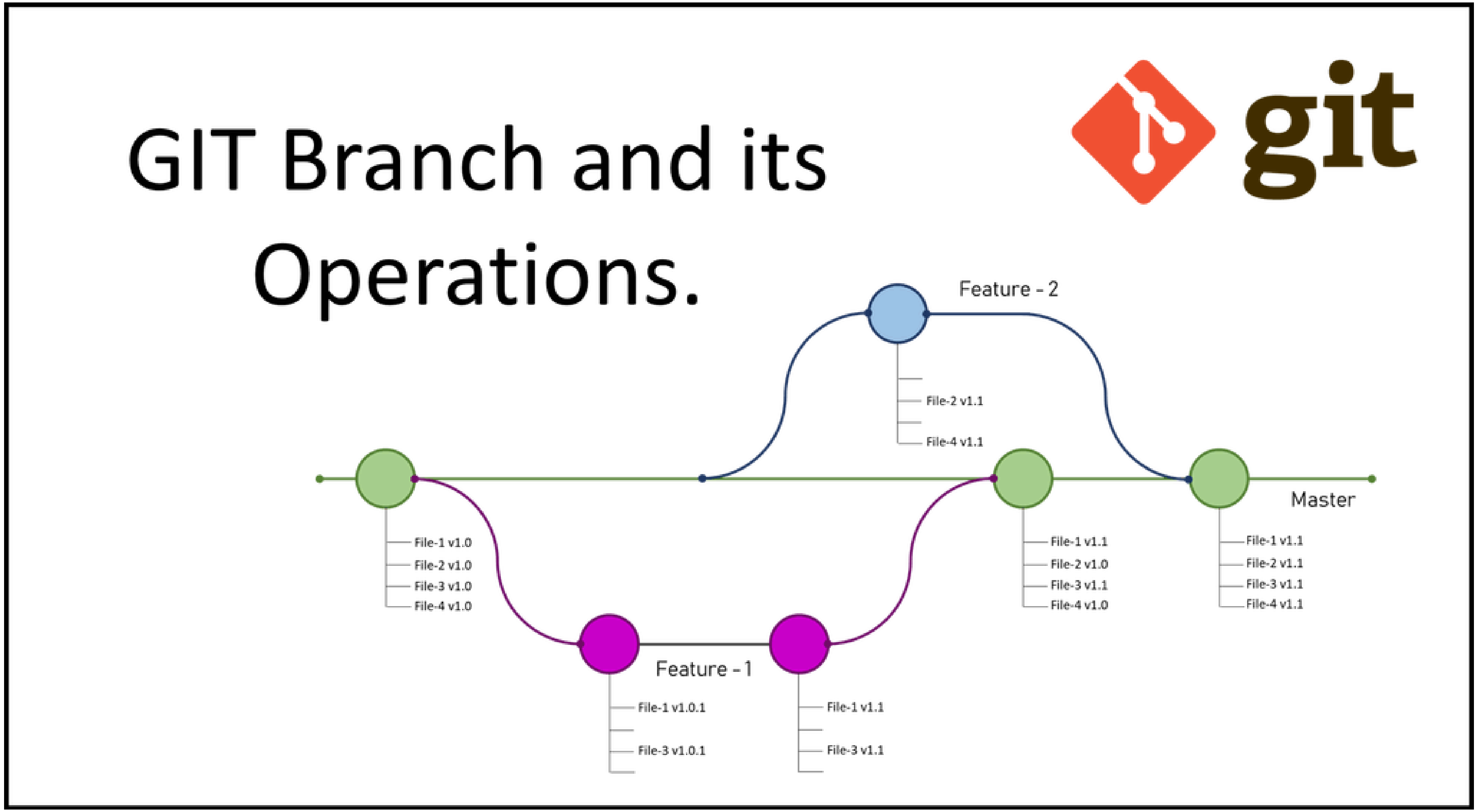
OUTLINE

- Brief overview
  - What it is
  - Why to use it
- What (not) to save
- Discussion

VERSION CONTROL

- Maintain different *versions* of your project, e.g.,
  - Stable
  - Development
- Maintain a *history* of your project
- Standard: git
- Note: `git` is **separate** from `github`
- `git` = MS Word, `github` = a shared google doc

VERSION CONTROL



VERSION CONTROL IS FOR *YOU*

Our goals:

1. Exactly reproducible
2. User friendly
3. Transparent
4. Reusable
5. Archived
6. Version controlled

Most of these are for *sharing* your analysis

Version control is mainly *for you*

IT’S WORTH IT

- Git is powerful, and so also fairly complex
- But the basic functionality is pretty simple
- It is well worth learning, and using regularly
- Many, many tutorials online
    - See, e.g., Karl Broman’s: https://kbroman.org/github_tutorial/

A SHORT TUTORIAL

GRAPHICAL INTERFACES

If you don’t like command-line interfaces, there are lots of graphics interfaces out there, e.g.

- GitHub Desktop
- GitKraken
- Sourcetree

WHAT (NOT) TO SAVE

AN INHERENT TRADE-OFF

You don’t normally save every file in your git repository.

- If you saved everything you did – every edit, every plot, etc. – your git archive would balloon in size
- This happens because git works off files differences (`diff`s)
- If you don’t save enough, you may not be able to piece together what you did

What should you keep?

Note:

- A `.gitignore` file lets you specify what (not) to save
- It is important to set this up at the outset of your project.
  Once a file has been committed to your repository, it generally can’t be removed.

GIT – DESIGNED FOR TEXT FILES

- When editing text files, git only saves *differences*
  This makes archiving simple text files very space efficient
- Other files get completely re-saved at each commit
- `git` is designed for code, not other assets (output, figures, data, …)

WHAT TO KEEP (TRACK)

- markdown files
- scripts
- makefiles
- simple text documentation
- etc.

WHAT TO IGNORE

- binary files
  - pdf, jpg, etc.
- data
- automatically generated text files
  - latex `.aux`, `.log`, etc.
- **notebook files that include output**
  - `.ipynb`
  - `.html`
- Generally, don’t track `.ipynb`
- **Use jupytext** to mirror to other formats
- e.g. `.R` or `.py` or quarto `.qmd` files are fine to track
- if you **really** want to track `.ipynb` then strip output using `nbstripout` beforehand

BUT DO SAVE YOUR NOTEBOOKS!

- Notebooks save code and output together (`.ipynb` or `.html`)
- It is *highly valuable* to archive notebooks (or some display version of them)
- Just not with every git commit!
- Develop some other strategy for saving notebooks,
  - e.g., a special directory where you put copies of “milestone” notebooks
  - e.g. some back-up scheme
- Include helpful archival information within your notebook, e.g.,
  - `date()`
  - `file.info(list.files(recursive=T))`
  - `installed.packages()`

DISCUSSION

- What strategy would work *for you* to archive “milestone notebooks”?
- Examples:
  - Have a special directory where you put notebooks (when?)
  - Instead of sending plots by email, send a notebook
- Realistically, what would be your biggest obstacle to actually following that strategy?
- What might you do to lessen that obstacle?