Durham University

COMP51915

attendance

Durham University

Version Control: Overview

COMP51915 – *Collaborative Software Development*
Michaelmas Term 2024

Christopher Marcotte[1]

[1] For errata and questions please contact christopher.marcotte@durham.ac.uk

Learning Goals

By the end of this workshop you should:

1. Understand the essentials of using version control *&* `git`
2. Understand the utility and mechanics of development collaboration
3. Be able to employ version control skills — especially for your project!

Version Control

Version control is a method of tracking changes to a directory containing subdirectories and files, over time, across many contributors.

In practice, version control is:

- a better way to manage changes to a codebase,
- a better way to collaborate than sending files, and
- a better way to share your code and other work with the world.

Why use Version Control?

Using a software tool to handle versioning of your project files frees you to work on the stuff that *matters* – solving problems.

Version control's advantages

- It's easy to set up – *we'll show you!*
- A git repository is a full backup of a project and its history
- Very few commands for most day-to-day version control tasks
- GitHub hosting provides a web-based collaboration service

Version Control Generally

Version Control can be understood as a Directed Acyclic Graph (DAG):

- whose *direction* is inherited from the flow of time (past → future)
- with *no cycles*, so future revisions are always distinct
- and a structure with potentially multiple trunks (product versions)

Version Control Generally

Version Control can be understood as a Directed Acyclic Graph (DAG):

- whose *direction* is inherited from the flow of time (past → future)
- with *no cycles*, so future revisions are always distinct
- and a structure with potentially multiple trunks (product versions)

In practice, it's sufficient to think of Version Control as a main trunk with *branches* and *merges*.

Branches are where an iteration points to more than one future iteration. .

Merges are where an iteration is pointed to by more than one past iteration.

Example: Version Control DAG

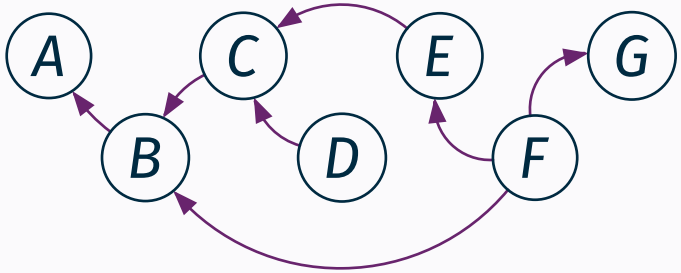

Figure 1: `Git-Graph` (amongst others) produces graph structures from git histories. Note: the arrows point *historically*, rather than *future*.

Example: Version Control DAG

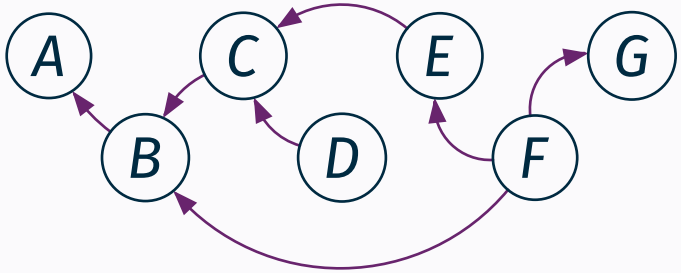

Figure 2: `Git-Graph` (amongst others) produces graph structures from git histories. Note: the arrows point *historically*, rather than *future*.

In this graph:

- Initial commit ***A*** updates to ***B***,
- Commit ***B*** produces ***C***,
- Commit ***C*** produces ***D***,
- Commit ***D*** is abandoned,
- The end-deliverable (***F***) has
  - early contributions (***B***),
  - late contributions (***E***), and
  - some from a *pull request*, (***G***)

Version Control Strategies

There are a number of branching strategies in `git` – details in a later lecture.

Figure 3: Git-graph'd history of the git-graph repository.

Version Control Strategies

There are a number of branching strategies in `git` – details in a later lecture.

Figure 5: Git-graph'd history of the git-graph repository.

Figure 6: Git-graph'd history of the Parallel Data Assimilation Framework repository.

Significant `git` diversions from other Version Control

1. `git` stores *snapshots* of the files in a directory, rather than a *base* file and accumulated *differences*
2.
3.
4.
5.

Significant `git` diversions from other Version Control

1. `git` stores *snapshots* of the files in a directory, rather than a *base* file and accumulated *differences*
2. Every operation is local-*first* – the remoteness of a repository is ancillary
3.
4.
5.

Significant `git` diversions from other Version Control

1. `git` stores *snapshots* of the files in a directory, rather than a *base* file and accumulated *differences*
2. Every operation is local-*first* – the remoteness of a repository is ancillary
3. `git` relies on *hashing* ensure integrity of the operations – no MITM
4.
5.

Significant `git` diversions from other Version Control

1. `git` stores *snapshots* of the files in a directory, rather than a *base* file and accumulated *differences*
2. Every operation is local-*first* – the remoteness of a repository is ancillary
3. `git` relies on *hashing* ensure integrity of the operations – no MITM
4. `git` generally *adds* data to a repository – the default actions are difficult to make *lose information*
5.

Significant `git` diversions from other Version Control

1. `git` stores *snapshots* of the files in a directory, rather than a *base* file and accumulated *differences*
2. Every operation is local-*first* – the remoteness of a repository is ancillary
3. `git` relies on *hashing* ensure integrity of the operations – no MITM
4. `git` generally *adds* data to a repository – the default actions are difficult to make *lose information*
5. There are *three* states for files in `git`: modified, staged, and committed; the staging step is where people tend to struggle

GitHub, GitLab, and `git` ting started

There are two dominant available `git` hosts – GitLab and GitHub.

They have their strengths and weaknesses – GitHub is more popular.

In order to use these services, we need to create an account.[2]

[2] The alternative is to create your own `git` server on your own hardware – this is called *self-hosting*. Doing so is not an especially technically difficult project, but it is probably not what you want to start with. Additionally, if your self-hosted repo becomes popular, it will substantially impact your server.

Creating an account

1. Navigate to github.com
2. Click the Sign up button in the upper-right corner
3. Enter an email address for your account (e.g., your Durham email)
4. Manage your login security[3]

[3] Including whether, why, and how to keep your email address private!

Creating an account

1. Navigate to github.com
2. Click the Sign up button in the upper-right corner
3. Enter an email address for your account (e.g., your Durham email)
4. Manage your login security[4]

Let's do that now.

[4] Including whether, why, and how to keep your email address private!

Creating a new repository on GitHub

The real utility of using GitHub over local `git` is storing and sharing code.

[5] You can also import an existing repository (though only `git` ones, now!) into GitHub for hosting.

Creating a new repository on GitHub

The real utility of using GitHub over local `git` is storing and sharing code.

Creating a new repository[6] requires:

1. Name (GitHub will offer you an available but nondescript – bad! – name)
   - Description of the repository content (concise!)
2. Select *Public* or *Private* – if other GitHub users can see it
3. Initialized with:
   - README (important information for a new user)
   - a `gitignore` file (language-dependent files that `git` will not track)
   - License (dependency-dependent!)

[6] You can also import an existing repository (though only `git` ones, now!) into GitHub for hosting.

GitHub and `git`

It is important to note that GitHub (or GitLab) and `git` are not the same entity.

`git` is an open-source program for version control which is based on a distributed repository model with cheap branching.

GitHub is a `git` host and website owned by Microsoft, which is leveraging its dominant market position to sell yours and many others work back to you through the white-washing of generative AI.

GitHub and `git`

It is important to note that GitHub (or GitLab) and `git` are not the same entity.

`git` is an open-source program for version control which is based on a distributed repository model with cheap branching.

GitHub is a `git` host and website owned by Microsoft, which is leveraging its dominant market position to sell yours and many others work back to you through the white-washing of generative AI.

I will speak mostly about `git`, rather than GitHub.

Installing `git` locally

To use `git` on the command line, you will need:

- A (`bash`) shell (and `ssh` for remote work)
- A `git` installation
- A basic text editor, like `nano`

[7] Use `sudo apt install git nano` on Linux, or use `brew install git nano` on macOS after installing Homebrew.

[8] You can install WSL from the command line with `wsl --install` or by using the Microsoft Store.

Installing `git` locally

To use `git` on the command line, you will need:

- A (`bash`) shell (and `ssh` for remote work)
- A `git` installation
- A basic text editor, like `nano`

On Linux and macOS, these should be pre-installed.[9]
On Windows, you should use the Windows Terminal.[10]

[9] Use `sudo apt install git nano` on Linux, or use `brew install git nano` on macOS after installing Homebrew.

[10] You can install WSL from the command line with `wsl --install` or by using the Microsoft Store.

Using Version Control Productively

Bad Practice

- Unclear Repository Names
- Intermittent commits
- Uninformative commit messages
- Useless README, no licensing

Good Practice

- Descriptive Repository Names
- Frequent commits
- Specific commit messages
- Documentation in repo

Using Version Control Productively

Bad Practice

- Unclear Repository Names
- Intermittent commits
- Uninformative commit messages
- Useless README, no licensing

Good Practice

- Descriptive Repository Names
- Frequent commits
- Specific commit messages
- Documentation in repo

Put in the effort *now* for easy maintenance in the *future*.

Collaboration using `git`

Publicly available `git` repositories are... *public*!

You should maintain standards of professionalism when publishing:

- Giving issues and pull-requests charitable understanding
- Using clear, concise, and professional language in your files
- Maintaining adequate documentation for reuse of your code

Collaboration using `git`

Publicly available `git` repositories are... *public*!

You should maintain standards of professionalism when publishing:

- Giving issues and pull-requests charitable understanding
- Using clear, concise, and professional language in your files
- Maintaining adequate documentation for reuse of your code

Your presence on GitHub is often something employers will look at!

Further Reading

1. The Software Carpentry git-novice tutorial covers the basics of `git`:
   - setting up and using `git`
   - collaborating with others using `git`
   - managing repository history and conflicts
   - the typical and essential use of `git` commands
2. There is also a ***free*** book on professional `git` usage.
3. *GitHub* publishes a series of documents called *Git Guide*

A word of advice

`git` is one of the most widely-used softwares in the world.

It is by developers, for developers, and it is a technical and deep topic.

That is: *if you find yourself frustrated with* `git` *– you are not alone!*

Someone else has run into the same issue, and worse.

Your first goal with `git` is to use it *productively* – not to use every feature.

If you only ever use `git add`, `git commit`, and `git push` – that's fine!

Rest of this Session

The rest of today is broken into three parts:

Getting Started

- making a new repository,
- navigating the GitHub website,
- and command line navigation.

Rest of this Session

The rest of today is broken into three parts:

Getting Started

- making a new repository,
- navigating the GitHub website,
- and command line navigation.

Maintenance

- branching & merging,
- branch management strategies,
- and responsible file management.

Rest of this Session

The rest of today is broken into three parts:

Getting Started

- making a new repository,
- navigating the GitHub website,
- and command line navigation.

Maintenance

- branching & merging,
- branch management strategies,
- and responsible file management.

Collaboration

- managing pull requests,
- managing teams, and
- distributed workflows

Rest of this Session

The rest of today is broken into three parts:

Getting Started

- making a new repository,
- navigating the GitHub website,
- and command line navigation.

Maintenance

- branching & merging,
- branch management strategies,
- and responsible file management.

Collaboration

- managing pull requests,
- managing teams, and
- distributed workflows

We expect you to follow along so you can use these skills in the programme.