Technical Report OUCS-2019-03

Git Badges: We’re not ready to commit yet

Authors:

Matthew Jenkin, Steven Mills, David Eyers

Department of Computer Science, University of Otago, New Zealand

Git Badges: We're not ready to `commit` yet

MATTHEW JENKIN, University of Otago
STEVEN MILLS, University of Otago
DAVID EYERS, University of Otago

ACM Reference Format:
Matthew Jenkin, Steven Mills, and David Eyers. 2019. Git Badges: We're not ready to `commit` yet. 1, 1 (November 2019), 24 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 ABSTRACT

Mastering the use of version-control systems (VCS) is a crucial skill for efficiently working with software. Undergraduate students should thus learn how to use a VCS early in their university studies, if they have not already done so. However, an introduction to the use of VCSs is not a good topic for academic teaching, when the VCS is just being used as a tool. *Micro-credentials* or *badges* show promise as a mechanism for lifting teaching of skills such as use of VCSs out of the flow of the academic curriculum, while supporting students appropriately. This paper presents our preparatory work to form a lesson on the `git` VCS, as part of a larger study into the use of a form of micro-credentials for skills-based teaching. We survey a number of popular, online resources for `git` education, contrasting their pedagogical approaches. We note key considerations for providing students a grounding in `git` skills that is efficient for their learning, at multiple levels of detail.

2 INTRODUCTION

Throughout their study, students need to acquire a number of different skills, and in some cases these skills do not fit naturally into a single part of the curriculum. In some situations, such as in a software tools module, these skills may be core to the subject at hand, but in others they are tangential — they are needed to complete coursework or otherwise engage with the material being taught. In the latter situation, skills-based material is often taught in a rather cursory manner, and students may encounter the same material repeatedly in different contexts.

We focus here on the example of Computer Science students learning a version-control system, specifically `git`. In our department, for example, `git` is used in papers in Web Application Development, Computer Game Design, Software Engineering, and Cloud Computing. Teaching and assessing students' mastery of `git` does not fit well within the department's broader curriculum structure for several reasons:

- While many papers use `git`, not all of them do so, and it would be inappropriate to displace material from the core teaching of programming (for example) to teach `git`.

Authors' addresses: Matthew JenkinUniversity of Otago, maffu@cs.otago.ac.nz; Steven MillsUniversity of Otago, steven@cs.otago.ac.nz; David EyersUniversity of Otago, dme@cs.otago.ac.nz.

- Mastering of `git` is a software skill, like using office productivity software, or online search tools, and so is a poor topic for academic teaching.

As a result, several papers rush though a basic `git` introduction which may be insufficient for more general application, or redundant for students who have already encountered the tool.

To improve this approach, we are investigating a system of *micro-credentials* to support skills-based learning. The term *micro-credentials* is used to refer to a wide range of units of teaching, learning, and assessment. The New Zealand Qualifications Authority (NZQA) defines them[1] as certifying "achievement of a coherent set of skills and knowledge" in a unit that is "smaller than a qualification". This makes micro-credentials 'small' in comparison to a degree or diploma, but they can still be larger than many university modules or papers. Micro-credentials are defined by the NZQA to be 5–40 points, whereas university papers are typically 15–20 points, and 120 points is a full-time course load for a year. We are interested in much smaller units of learning than this, and so use the term *badge* to distinguish our approach from the NZQA framework [18]. The term *badge* also has an informal tone, and suggests a threshold of achievement rather than a graded piece of work, which is appropriate to our goals.

Our approach is being piloted with `git`, but we can see scope for extension to other skills-based technologies such as use of LaTeX for document preparation, or `Docker` for containerisation. In this paper we describe our progress so far, and give some initial indications of its value to students. Our approach is to encourage self-directed study, to use existing resources such as online tutorials, and to give students the ability to explore further than is required for their immediate needs. We present the material we have found for teaching `git` in some detail for two reasons:

- Firstly, this allows us to identify several considerations that we believe generalise to most other skills-based technologies in computer science.
- Secondly, we have identified several resources that may be of specific interest to others wishing to give students a background in `git` in different contexts.

We have conducted a preliminary trial of this approach, and received student feedback. While this trial was not large enough to produce conclusive results, or to support statistical analysis, it suggests that both `git` itself, and this approach were useful to students, and there are indications that they use the skills they have gained outside of the context in which they were initially taught.

3 LEARNING GIT

There are a vast number of resources available for learning `git`, so our search was restricted to those that were recommended on the official `git` website[2]; were on the first page when googling "*learn git*", "*git tutorial*", or "*git workflow*"; or were recommended by one of the previously-examined resources. We further restricted the pool by eliminating resources of impractical length (*e.g.* online books) and resources that required user registration (*e.g.* "**Codecademy**"[3]).

In order to determine what needs to be taught, and how it should be approached, one needs to consider the prospective learner:

- What are their goals? (*e.g.* Will they use `git` for personal projects, or collaborate?)
- What prior experience do they have with the topic? (*e.g.* Have they used `git` before?)

[1]https://www.nzqa.govt.nz/providers-partners/approval-accreditation-and-registration/micro-credentials/
[2]http://git-scm.com/
[3]http://www.codecademy.com/

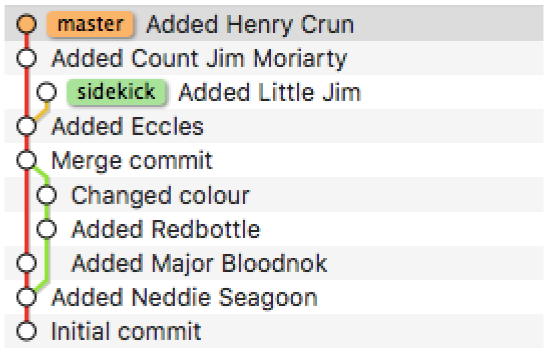

Fig. 1. An example of GitX's display of a fictional repository's commit history. The 'master' branch is currently checked-out, and there is another branch called 'sidekick'. A previous branch has been merged into master, and thus no longer needs a separate label.

- Do they have experience with a related topic? (*e.g.* Have they used other version control systems before?)
- What sort of interface are they familiar with? (*e.g.* Have they used a command-line shell before?)
- Are they familiar with relevant theoretical concepts? (*e.g.* Do they know anything about graph theory?)

Our assumed learners are Computer Science students (and thus are either familiar with, or at least have a context for, data structures like a DAG[4]), they are not expected to have any previous version control experience, they will likely be working on both individual and group projects, and the assumed interaction with `git` is via a `Unix`-like command-line interface. Resources focused on a different interface (*e.g.* many IDEs[5] integrate `git`, making most of the everyday `git` operations simpler, or even automated) were omitted from the list, unless they also covered command-line interactions.

As a common addition to the command line, many of the resources also use a basic GUI[6] like **`gitk`** (commonly distributed with `git`), **`gitg`** (a `linux`-specific clone of `gitk`), or **`GitX`** (an `OSX`-specific clone of `gitk`). All three show the commit tree in the same manner as the "`git log --graph`" command, *i.e.* with undirected links, branches and tags labelled (but not showing the `HEAD` label), and time progressing bottom-up (see Figure 1).

4 ISSUES NOTED

The obvious point to start with is that quality of presentation is not correlated with quality of content. Some tutorials are presented very neatly and consistently, but often describe elements in a confusing, misleading, or incorrect way. Even `GitHub`[7] (the most widely-used hosting/development provider for `git` repositories) and `GitLab`[8] present the commit history incorrectly[9], that is, parents point to children. `BitBucket`[10] and `SourceForge`[11], who round out the top four[12], present the commit history with undirected links (similar to `gitk`'s display). Presumably the purpose of these diagrams is to show changes over time, which is helpful from a user's perspective, but unhelpful from a *learner's*

[4]Directed Acyclic Graph; `git`'s commit history is stored as a DAG, and many operations are defined by how they effect it.
[5]Integrated Development Environment.
[6]Graphical User Interface.
[7]http://github.com/
[8]http://gitlab.com/
[9]Displayed via the "`Insights>Network`" menu option (which specifically shows forks/pull-requests, so technically is a subset of the commit history) on `GitHub`, or the "`Repository>Graph`" menu option on `GitLab`
[10]http://bitbucket.org/
[11]http://sourceforge.net/
[12]See http://en.wikipedia.org/wiki/Comparison_of_source-code-hosting_facilities#Popularity

perspective. An awareness of accessibility in the commit DAG helps with understanding fast-forward merges (one commit is in the history of the other), and why having a detached `HEAD` is a problem (any commits created will be unreferenced, and thus not (easily) accessible).

The resources vary in their tone; some are more informal and chatty, others more formal and structured. While most of the underlying concepts and details can be definitely right or wrong (*e.g.* typing '`git add myfile.txt`' has a specific effect), higher-level topics like workflows and conventions are not so clear cut, and thus it is appropriate to describe them more as suggestions than injunctions. Unfortunately, people can be dogmatic about their preferences. "GitFlow considered harmful"[31], for example, has a very closed-minded tone (especially in responses to comments). In reality, different workflows and conventions will suit different kinds of project, and different team dynamics.

Some inaccuracies of detail could be charitably interpreted as "lies to children" (*i.e.* abstractions or simplifications[13] that are easier to understand as a learner[14]), though the resources seldom indicate that they are making such a simplification. For example "Ry's Git Tutorial"[see 16, page 53] states that you *cannot* make a commit in a detached `HEAD` state (you can, but the new commit will be unreferenced — see above — and liable to be garbage-collected, which is a whole other topic), and "Version Control for Designers"[see 13, section "Branching"] equates branching/merging with cloning/pushing (correct in the abstract sense, in that both are about separating/reintegrating a parallel development, though they have very different effects and consequences).

Others are harder to justify. For example "Git and GitHub Basics" on "Trailhead"[see 33, section "Learn Where GitHub Fits in the Development Lifecycle"] defines "merge" as "The combined history of two or more branches", which is expressed simply, and technically correct, but is defining the *noun* form (the resulting state), when "merge" is most often used as a *verb*, and the action of merging is far more important for a `git` user to understand. Reading other material (even later parts of the same resource) will be more confusing if the learner is thinking about a shared history rather than how to amalgamate the contents of two branches. As another example of an inadvisable simplification, the `git` tutorial on "Tutorials Point"[see 44, section "Basic Concepts>DVCS Terminologies"] describes the commit history as a linked-list[15] (which implies there are no other branches) though they mention merge commits as having more than one parent (which implies there *are* other branches).

The topics covered should advance sensibly, each building off the one(s) that came before, whether that ordering is based upon concepts (*i.e* explaining how/why `git` behaves as it does, covering progressively more complex/dangerous actions), workflows (*i.e.* creating a repository, adding & committing files, looking back at a previous version, ...), or some other ordering (*e.g.* alphabetically for a command-reference). Regardless, a resource should cover all the necessary and sufficient topics to enable the learner to use `git` effectively (for example, as stated previously, our assumed learners require `git` for both individual and group projects, so a resource that does not cover interacting with shared remote repositories would be insufficient).

The ease of branching and merging is often touted as one of the strengths of `git`, yet some resources omit those operations. "Version Control with Git" on "Software Carpentry"[see 40, section "Conflicts"], for example, advises making frequent use of branches, but does not cover what they are or how to use them[16]. Many others show branching and merging, but don't address merge conflicts ("Git Happens"[see 21, from approx. 43:48] explicitly calls out other tutorials for ignoring merge conflicts). Even some systems built on top of `git` don't address conflicts as well as they could;

[13]They should not be outright lies; the learner should be able to use the concept as a starting point, adding detail and nuance as they learn more about the topic.
[14]Ideally it should be clear to the learner that any distortion is an intentional simplification, lest they lose faith in the teacher.
[15]Another common graph-like data-structure, where the nodes are connected in a single, linear path.
[16]For such a high-profile and well-regarded resource provider, this seems a particularly glaring omission.

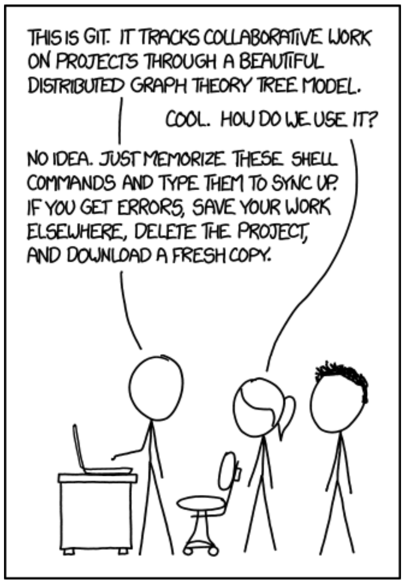
THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Fig. 2. "If that doesn't fix it, git.txt contains the phone number of a friend of mine who understands git. Just wait through a few minutes of 'It's really pretty simple, just think of branches as...' and eventually you'll learn the commands that will fix everything." — from https://xkcd.com/1597/

`GitHub` workflows frequently suggest dealing with merge conflicts in your local `git` repository then pushing the result, as `GitHub`'s interface can only handle line conflicts (*e.g.* trying to merge two different versions of the same line in a file)[17]. When discussing merge conflicts, it is also important to note that any other operation that can copy a commit from one branch to another (*e.g.* `cherry-pick`, `rebase`) has the possibility of conflicts (which are resolved via much the same process as merge conflicts). This is often not mentioned, implying to the learner that these other operations are somehow "cleaner", or less likely to cause problems than `merge`.

Unless a resource is intended as a command reference, and not a learning tool, then some explanation should be given for the commands and options used. Some resources treat particular commands like mystical incantations — whether covering everyday workflows, or how to solve problems, the user is just given a set of commands to enter (satirised in Figure 2). Knowing even a little about what those commands are doing will make them easier to remember (or to search for, if they have been forgotten), and better enable the user to adapt to other scenarios. An explanation is also important if the command has particularly drastic effects (*i.e.* that are difficult, or even impossible, to undo); for example, an operation like `reset` can result in unreachable commits[18]. A particularly important warning is that no operations that "rewrite history" should be used once that history has been shared, as later updates will become messy and complicated (unless *every other user* forcibly updates their own repositories to match the rewritten history).

5 TOPIC MATRIX

Each topic related to `git` can be thought of (from a teaching perspective) as having two attributes:

[17]To be fair, this probably covers the majority of conflicts; about the only other type of conflict we've seen is if one branch has moved/modified a file and another has deleted the same file.

[18]Whether the unreachable commits are presented as "gone forever" or "not easy to get back, and eventually will be deleted" depends on how much is being simplified at that stage.

- **Complexity/Level** — Does this topic belong in a Beginner, Intermediate, or Advanced tutorial? *e.g.* how to use the `add` and `commit` commands is one of the first things a `git` user needs to know; interactive rebase (`rebase -i`) is best introduced to someone who already knows something about how `git` works.
- **Importance** — Is this required, helpful, or merely interesting? *e.g.* understanding how to use `push` and `fetch` is necessary for collaboration; knowing what attributes affect a commit's `SHA-1` hash and why that makes comparisons easier and networking more efficient doesn't usually impact the way you work.

Topics in the table (shown below) are listed alphabetically (not including '`git`' at the start of every command, so for example `git status` is treated as 'status'). Most commands have additional options (not listed here) that may be considered "Advanced" (for example, `git rebase`'s `--onto` option).

Topic	**Level**	**Importance**

`[$] git add FILE`
`[$] git add .`
	Beginner	Essential

Advanced staging (between files, or within files)	Intermediate	Interesting
`[$] git am FILE/DIRECTORY`	Intermediate	Useful
`[$] git apply`[19] `PATCH`	Intermediate	Useful
`[$] git blame FILE`	Intermediate	Useful
`[$] git branch . . .`	Intermediate	Essential
`[$] git checkout COMMIT`	Beginner	Essential
`[$] git checkout -b COMMIT`	Intermediate	Useful
`[$] git cherry-pick COMMIT`	Intermediate	Essential
`[$] git clean . . .`	Intermediate	Useful
`[$] git clone REPOSITORY [DIRECTORY]`	Intermediate	Essential
`[$] git commit –amend . . .`	Intermediate	Useful
`[$] git commit -m MESSAGE`	Beginner	Essential

[19]All the other commands mentioned in this table are listed as "porcelain" commands by the official `git` documentation; this one is listed as a "plumbing" command (*i.e.* not part of the normal user interface).

Topic	**Level**	**Importance**
Commit (and other object) IDs (`SHA-1` hash, unique 40-character hexadecimal string)	Beginner	Essential
Commit ID benefits (checksum[20] of commit; includes author, timestamp, snapshot of files, parent reference(s); allows for quick comparisons)	Beginner	Interesting
Commit messages should be meaningful (formatting convention of summary, blank line, explanation)	Beginner	Useful
Commit-ish parameters (tag, branch, relative reference, *etc.*)	Intermediate	Essential
Commit tree (pointers to parent(s), effects of operations)	Intermediate	Essential
`[$] git config . . .`	Beginner	Useful
Dangling commits, how to retrieve them (*e.g.* using the `-reflog` option for `git log`), garbage collection[21]	Intermediate	Useful

`[$] git diff`
`[$] git diff -staged/-cached [COMMIT]`
`[$] git diff COMMIT`
`[$] git diff COMMIT COMMIT`
	Beginner	Useful

`[$] git fetch [REMOTE]`	Intermediate	Essential
`[$] git format-patch COMMIT`	Intermediate	Useful
`[$] git gc . . .`	Intermediate	Interesting
`[$] git help COMMAND`	Beginner	Useful
Ignoring files you don't want to store in the repository (using `.gitignore`)	Beginner	Essential
`[$] git init`	Beginner	Essential

[20]If the contents have changed, the generated ID will be different.
[21]It is worth being aware of `git`'s approach to garbage collection of unreferenced commits, but this does not require knowledge of the `git gc` command.

Topic	**Level**	**Importance**
`[$] git log`	Beginner	Essential
`[$] git log ...`	Beginner	Useful
`[$] git merge ...`	Intermediate	Essential
Merge conflicts (also from `rebase`, `cherry-pick`, `revert`, *etc.*)	Intermediate	Essential
Merge/rebase controversy — each suited to different situations	Intermediate	Useful
Motivation for version control (and distributed version control in particular)	Beginner	Interesting
`[$] git pull [-rebase] [REMOTE [BRANCH]]`	Intermediate	Useful
`[$] git push [REMOTE [BRANCH]]`	Intermediate	Essential
`[$] git rebase DESTINATION [SOURCE]`	Intermediate	Essential
`[$] git rebase -i/-interactive DESTINATION [SOURCE]`	Intermediate	Useful
Relative references (`~N` and `^N`)	Intermediate	Useful
`[$] git remote ...`	Intermediate	Useful
`[$] git request-pull ...`	Intermediate	Useful
`[$] git reset [FILE]`	Intermediate	Essential
`[$] git reset [-soft/-mixed/-hard] COMMIT`	Intermediate	Essential
`[$] git revert COMMIT`	Beginner	Essential
"Rewriting" history is actually *replacing* history	Intermediate	Useful
Shared history should not be "rewritten", causes problems for other users/repositories	Intermediate	Essential

Topic	Level	Importance
`[$] git shortlog . . .`	Intermediate	Interesting
`[$] git show . . .`	Intermediate	Interesting
`[$] git stash . . .`	Intermediate	Useful
`[$] git status`	Beginner	Essential
Stores commits as a snapshot of working directory; many operations work with the differences between those snapshots	Beginner	Useful
`[$] git tag NAME [COMMIT]`	Beginner	Essential
Telling a project's story via its commit history (logical — focused on features, or chronological — focused on timestamps)	Intermediate	Interesting
Three areas (working directory, staging area, commit tree)	Beginner	Essential
Uncommitted changes can be lost by doing other operations that affect the working directory	Beginner	Useful
Used by many other systems (e.g. `GitHub`, integrated into many IDEs[22])	Beginner	Interesting

6 TUTORIAL COMPARISON

Findings from comparing the available resources are presented in the table below. While the first few column headings should be relatively self-explanatory, the meaning of each is provided.

- **Resource** — The title of the resource (each of which is cited in the bibliography).
- **Format** — Is this resource text, audio, video, images/ diagrams?
- **Audience** — Is this resource aimed at people who have never used `git` before? Or people who have used `git` but want to understand it better? Are they expected to be familiar with any other technologies?
- **Purpose** — Is this resource intended to be a reference? To explain how to perform various actions with `git`? To explain why `git` works the way it does?

[22]Integrated Development Environment.

- **Focus** — What part of `git` is the resource covering? As with any substantial system, `git` can be described from several different perspectives. Some resources are more general, touching on a wide range of `git`'s features. Others are more specific, dealing only with a particular aspect.
- **Three Areas** — Does the resource distinguish between the working directory (files that are accessed normally), the staging area/index/cache (where commits are prepared), and the commit history (see 'Commit Tree' below)? This distinction is important in order to understand the way many `git` commands operate. Making staging and committing a two-step process allows greater control over what changes are included in a particular commit. Even if you are not using advanced staging, when something goes wrong, it is helpful to understand the state of each of the three areas in order to resolve the issue.
- **Commit Tree** — How well does the resource cover the commit history? A lot of `git` operations affect mainly this third area; manipulating the DAG that represents the commits stored in the repository. Every location in the graph[23] represents a commit, and each commit has a reference to its parent(s). Each branch is a pointer to a particular commit (the current 'tip' of that branch). Visualising the result of these connections helps with understanding operations like merge and rebase. Connections being only one-way (*i.e.* child⟶parent) makes fast-forward merges clear. `HEAD` represents the currently checked-out commit, and usually points to a branch (`HEAD` pointing directly to a commit is what `git` calls 'detached `HEAD` state'). As "Think Like (a) Git" puts succinctly, "references make commits reachable"[see 24, section "Experimenting With Git"]. If a commit does not have any children, any branch pointers, or any tags, then it is unreachable, and will be deleted when `git` does its garbage collection[24].
- **Remotes** — Does the resource cover remote repositories? A single-user project can involve a remote repository, but it effectively functions as just an off-site backup; in order to use `git` collaboratively, separate repositories must interact. This can often create confusion, due to having to integrate changes from different people (which may contradict each other), and the different repositories needing to push/fetch regularly to stay in sync.
- **Conflicts** — Does the resource acknowledge that conflicts can result from operations like `merge`? Does it show how to deal with those conflicts? Several other operations can also lead to similar conflicts (*e.g.* `rebase`, `cherry-pick`), but merging is usually where conflicts are first encountered. Understanding the conflict resolution process prevents the user panicking when a conflict arises, and allows them to focus on the content rather than the process. There are different tools that can help with resolving conflicts, but are typically designed for a particular type of conflict, so are not useful in all cases.
- **Workflow** — Does the resource describe any `git` workflows? What type? In how much detail? While any workflow must be tailored to the particular project (*e.g.* Will a single release, or multiple concurrent releases be supported?), and the particular team (*e.g.* How many team members are there? Is there a team hierarchy?), there are several general patterns/approaches that are widely used. There is less of a "right answer" here; this column is more about recognising whether the resource has made any recommendations.

[23]It is often referred to as a tree, but technically it is a graph.
[24]Regular automated tidying of `git`'s internal storage.

Resource	Format	Audience	Purpose	Focus	3 Areas	Commit Tree	Remotes	Conflicts	Workflow
Merge or Rebase?[1]	Essay, screenshots	Intermediate	Why-To	Collaboration workflow	No	Only implicitly (discusses merging/rebasing)	Only implicitly (discusses sharing)	No	Overview
Smart branching with SourceTree and Git-flow[2]	Essay, screenshots	Advanced	How-To	Collaboration workflow	No	Only implicitly (discusses branching/merging)	No	No	Overview
Deliberate Git[3]	Talk transcript[25], command-line, screenshots	Advanced	Why-To	Commit messages	No	Only implicitly (discusses rebasing)	No	No	Basic
Git Masterclass[4]	Video, exercises, diagrams, command-line[26]	Beginner–Advanced, expects some familiarity with version control	How-To	Specific commands	Yes, diagram	Yes, undirected labelled (but without `HEAD`) left-to-right	Yes	Only implicitly (`merge`, `pull`, `rebase`, `cherry-pick`)	Overview
5 types of Git workflow that will help you deliver better code[5]	Essay, diagrams	Intermediate	Overview	Workflows	No	Yes, undirected labelled (without `HEAD`) left-to-right	Yes	No	Yes

[25]Slides also available.
[26]Slides and exercises also available.

Resource	Format	Audience	Purpose	Focus	3 Areas	Commit Tree	Remotes	Conflicts	Workflow
CakeDC Git Workflow[6]	Essay, diagrams, command-line	Advanced	How-To	Workflow	No	Yes, sparse[27] wrongly-directed spatially-labelled (without `HEAD`) top-down	No	No	Yes
GitHub Flow[7]	Essay, command-line, screenshots	Advanced	How-To	Workflows	No	Only implicitly (discusses branching/merging)	Yes	No	Yes
Git Workflows That Work[8]	Essay, diagrams	Intermediate	Overview	Workflows	No	Yes, wrongly-directed labelled[28] bottom-up	Mentioned (discusses sharing)	Mentioned (`merge`)	Yes
Learn Git Branching[9]	Interactive, command-line, tree diagrams	Intermediate–Advanced	How-To, Sandbox	Specific commands, effects on commit tree	No	Yes, directed labelled[29] top-down[30]	Yes	No	No

[27]Diagrams are showing the movement of code during the workflow, so only show key commits, *e.g.* where one branch was merged into another.
[28]General branch indications are given via spatial labels, *e.g.* feature branches on the right.
[29]Only shows `HEAD` if detached, marks current branch label with $*$ otherwise.
[30]Branches are coloured, but the colours seem to be randomly chosen so are not always visually distinct.

Resource	**Format**	**Audience**	**Purpose**	**Focus**	**3 Areas**	**Commit Tree**	**Remotes**	**Conflicts**	**Workflow**
A successful Git branching model[10]	Essay, diagrams, command-line	Advanced	How-To	Workflow	No	Yes, wrongly-directed spatially-labelled[31] (without `HEAD`) top-down	Yes	Mentioned (`merge`)	Yes
git — the simple guide[11]	Command reference, diagrams	Intermediate	How-To	Specific commands	Yes, diagram[32]	Yes, but only abstract	Single-user, implied collaboration	Yes (`merge`)	No
Git Is Simpler Than You Think[12]	Essay, command-line	Intermediate, auto-didacts[33] with programming experience	Demystifying	Effects on internal files	No	Only implicitly (talks about commits pointing to their parents)	Mentioned (talks about sharing)	No	No
Version Control for Designers[13]	Essay, command-line, diagrams	Beginner	How-To	Basic commands	No	Yes, `gitk` output	Yes	No	Basic
A stable mainline branching model for Git[14]	Essay, diagrams, command-line	Advanced	How-To	Workflow	No	Yes, wrongly-directed spatially-labelled (without `HEAD`) bottom-up	Mentioned (discusses sharing)	Mentioned (`merge`, `rebase`)	Yes

[31]Notes tags with speech-bubble-like markers attached to commits, but these are nearly indistinguishable from meta-markers used, for example, to indicate the purpose of different branches.
[32]Refers to the commit tree as "HEAD".
[33]Presents a very dangerous do-as-I-do example: "I just looked online until I found a command that sounded like it did what *I would do myself* if I had to write some code to poke around the object database".

Resource	Format	Audience	Purpose	Focus	3 Areas	Commit Tree	Remotes	Conflicts	Workflow
Practical Git: A Workflow to Preserve Your Sanity[15]	Essay, command-line, diagrams	Intermediate/ Advanced	How-To	General commands, workflows	No	Yes, undirected spatially-labelled left-to-right	Yes	Mentioned (`merge`)	Basic
Ry's Git Tutorial[16]	Essays, diagrams, command-line[34]	Beginner–Advanced	How-and-Why-To	Specific commands, effects on commit tree	Yes, diagram	Yes, directed labelled (without `HEAD`)[35] left-to-right[36]	Yes	Yes (`merge`)	Yes
Git Succinctly[17]	Essays, diagrams, command-line[37]	Beginner–Intermediate	How-and-Why-To	Specific commands, effects on commit tree	Yes, diagram	Yes, directed labelled (without `HEAD`)[35] left-to-right[36]	Yes	Yes (`merge`, `pull`)	Yes
Git Workflows for Pros: A Good Git Guide[19]	Essay, diagrams	Advanced	Why-To	Workflows	No	Yes, sparse[27] wrongly-directed spatially-labelled (without `HEAD`) top-down	Mentioned	Mentioned (`merge`)	Overview

[34]Published as an e-book (`epub`); an online version exists, but is only available through the Internet Archive, so some images and linked files are missing.
[35]The currently checked-out commit is highlighted, but this does not distinguish where `HEAD` is attached.
[36]Gradually adds features to the commit tree diagrams as they are introduced in the text.
[37]Published as an e-book (`pdf`/`mobi`/`epub`); an online version is also available.

Resource	Format	Audience	Purpose	Focus	3 Areas	Commit Tree	Remotes	Conflicts	Workflow
A succesful Git branching model considered harmful[20]	Essay, diagrams	Advanced	Injunction	Workflow	No	Yes, undirected[38] labelled (without `HEAD`) top-down/left-to-right	Yes	Mentioned (`merge`)	Yes
Git Happens[21]	Video, diagrams	Beginner, assumes some version control experience	Demystifying, Why-To	Inner workings, basic commands	Yes, diagram	Yes, directed labelled bottom-up	Yes	Yes (`merge`, `pull`)	Basic
An Automated Git Branching Flow[22]	Essay, diagrams	Advanced	How-To	Workflow	No	Yes, wrongly-directed spatially-labelled (without `HEAD`) top-down	Mentioned (discusses sharing)	Mentioned (`merge`, `rebase`)	Yes
5 Git workflow best practices you've got to use[23]	Essay, command-line	Intermediate	Advice	Workflows	Only implicitly (discusses advanced staging)	Only implicitly (discusses merging/rebasing)	No	Mentioned (`merge`, `rebase`)	Basic

[38]There are some arrows in one diagram, but they indicate cherry-picking.

Resource	Format	Audience	Purpose	Focus	3 Areas	Commit Tree	Remotes	Conflicts	Workflow
Think Like (a) Git[24]	Essay, diagrams, command-line	Intermediate	Demystifying	Graph manipulation, some commands	No	Yes, sometimes `GitX` screenshots, sometimes directed labelled (without `HEAD`) left-to-right	No	No	No
A Visual Git Reference[25]	Command reference, diagrams	Intermediate	How-To	Specific commands, effects on commit tree	Yes, diagram	Yes, directed labelled left-to-right	No	No	No
David Mahler — Introduction to Git[26]	Videos (3), diagrams, command-line	Beginner	How-To	Specific commands	Yes, diagram	Yes, undirected labelled bottom-up	Yes	Yes (`merge`)	Yes
Backlog — Git Tutorial[27]	Essays, diagrams, screenshots, command-line	Beginner–Intermediate	How-To[39]	General, specific user interfaces[40]	Yes, diagram	Yes, (un)directed (un)labelled left-to-right[41]	Yes	Yes (`merge`, `pull`, `rebase`, `cherry-pick`)	Yes

[39] First two sections are mostly conceptual, though some (relatively advanced) commands are mentioned (*e.g.* `rebase`); third section introduces command specifics.
[40] Covers `TortoiseGit` for `Windows`, `SourceTree` for `OSX`, and command-line.
[41] Presents commit-tree diagrams very inconsistently. Often the edges are undirected; those that do have arrows are pointing the wrong way (*i.e.* parent to child). Labels are not always present, and are occasionally displayed differently. Meta-notes (*e.g.* "commit to be undone") are indistinguishable from branch labels.

Resource	Format	Audience	Purpose	Focus	3 Areas	Commit Tree	Remotes	Conflicts	Workflow
Coding Blocks — Comparing Git Workflows[28]	Podcast, online notes	Advanced	Why-To	Workflows	No	Only implicitly (discusses branching/merging)	Yes	Mentioned (`merge`, `pull`, `rebase`, `cherry-pick`)	Yes
4 branching workflows for Git[29]	Essay	Advanced	Why-To	Workflows	No	Only implicitly (discusses branching/merging)	Only implicitly (mentions sharing)	No	Yes
The Git Parable[30]	Essay	Beginner	Demystifying	General background	Yes	Yes, but only described	Only implicitly (discusses sharing)	No	No
GitFlow considered harmful[31]	Essay, diagrams	Advanced	Injunction	Workflow	No	Yes, undirected labelled (without `HEAD`) bottom-up	No	Mentioned (`merge`)	Yes
OneFlow - a Git branching model and workflow[32]	Essay, command-line, diagrams	Advanced	How-To	Workflow	No	Yes, wrongly-directed spatially-labelled (without `HEAD`) top-down	Mentioned (discusses sharing feature branches)	No	Yes

Resource	Format	Audience	Purpose	Focus	3 Areas	Commit Tree	Remotes	Conflicts	Workflow
Trailhead — Git and GitHub Basics[33]	Essays, diagrams, screen-shots, command-line	Beginner, designed for employees of the `Salesforce` company[42]	Feature listing[43]	Workflows, specific commands[44]	Yes, diagram	Yes, directed labelled[45] left-to-right, and example of "`git log`" (decorated graph) output	Yes	Yes (`merge`)	Small group, `GitHub`-specific
Understanding the Git Workflow[34]	Essay	Intermediate	Why-To	Workflows	No	Only implicitly (discusses merging/rebasing)	No	No	Yes
Corey Schafer — Git Tutorial for Beginners[35]	Videos, diagrams, command-line	Beginner	How-To	Specific commands	Yes, diagram	No	Yes	No	Small group
Corey Schafer — Git Tutorials[36]	Videos (5), command-line	Intermediate–Advanced	Problem solving	Specific commands	Yes	No	No	Yes (`merge`)	No
git-rebase[37]	Essay	Advanced	Why-To	History management	No	Only implicitly (discusses merging/rebasing)	Only implicitly (mentions pushing/pulling, pull requests, patches)	Mentioned (`merge`, `rebase`)	Overview

[42]A login is required in order to submit the quizzes at end of each unit and earn a "badge", the material is otherwise publicly available.
[43]Seems very marketing-focused, and has frequent unexplained acronyms and jargon.
[44]Far more focused around describing what *can* be done than explaining *how* to do it (commands are treated like mystic incantations) or *why* you would want to.
[45]Labels are just placed alongside commits, so it's not clear what they are attached to (*i.e.* when a commit has multiple labels).

Resource	Format	Audience	Purpose	Focus	3 Areas	Commit Tree	Remotes	Conflicts	Workflow
Git For Ages 4 And Up[38]	Video, physical models, command-line	Intermediate	Demystifying	Specific commands, effects on commit tree	No[46]	Yes, undirected[47] labelled (`HEAD` attached to commit) bottom-up	Yes	No	Basic
Thou Shalt Not Lie: git rebase, amend, squash, and other lies[39]	Essay	Advanced	Injunction	History management	No	Only implicitly (discusses merging/rebasing)	Only implicitly (mentions `push`)	No	No
Software Carpentry — Version Control with Git[40]	Command-line, exercises, diagrams[48]	Beginner, doesn't assume much command-line experience	How-and-Why-To	Specific commands	Yes, diagram	No	Yes	Yes (`pull`, by implication `merge`)	Small group
Visualising Git[41]	Interactive, command-line, tree diagrams	Intermediate	Sandbox, a few How-To examples	Specific commands, effects on commit tree	No	Yes, directed labelled[45] left-to-right	Yes, multi-user implied	No	No
The Thing About Git[42]	Essay, command-line	Advanced	How-and-Why-To	Advanced staging	Yes	No	No	No	No

[46]Does distinguish between "staged" and "committed", and mentions advanced staging, but is not explicit about the staging area.
[47]Later in the video it it noted that all connections implicitly point "down" (*i.e.* child to parent(s)).
[48]Diagrams often a bit over-complicated and hard to interpret.

Resource	**Format**	**Audience**	**Purpose**	**Focus**	**3 Areas**	**Commit Tree**	**Remotes**	**Conflicts**	**Workflow**
Re: [git pull] drm-next[43]	Email	Advanced	Injunction[49]	History management	No	Only implicitly (discusses merging/rebasing)	Only implicitly (mentions sharing)	No	Basic
Tutorials Point — Git Tutorial[44]	Essays, command-line, diagrams	Beginner[50]	How-and-Why-To	Concepts, specific commands	Yes, diagram[51]	Yes[51], directed labelled (without `HEAD`) left-to-right	Yes	Yes (`pull`, by implication `merge`)	Small group
Learn the Basics of Git in Under 10 Minutes[45]	Essay, command-line, diagrams	Beginner	How-To	Basic commands	Yes, diagram	No, but does show "`git log`" (non-graph) output	Yes	No	No
Git For Computer Scientists[46]	Essay, diagrams	Intermediate, assumes understanding of Computer Science concepts	Demystifying	Inner workings	No	Yes, directed labelled (without `HEAD`) bottom-up	Yes, multi-user implied	No	No
Explain Git With D3[47]	Interactive, command-line, tree diagrams	Intermediate	Sandbox, a few How-To examples	Specific commands, effects on commit tree	No	Yes, directed labelled[45] left-to-right	Single-user	No	No

[49]Fairly dogmatic, which he's upfront about: "You're very much allowed to quote me on this and use it as an explanation of 'do this, because that is what Linus expects from the end result'."[43] Other resources advocate a similar position, but give a more reasoned argument for it.
[50]Says no prior version control experience is expected, yet often likens commands to their `Subversion` equivalents.
[51]Diagrams are given, but not explained in the text.

Resource	Format	Audience	Purpose	Focus	3 Areas	Commit Tree	Remotes	Conflicts	Workflow
Git Immersion[48]	Command-line, exercises	Beginner–Intermediate	How-To	Specific commands	Only incidentally (talks about staging control)	Yes, (customised) "`git log –graph`" output	Yes	Yes (`merge`)	No
Git Workflow Explained — A Step-by-Step Guide[49]	Essay, diagrams, command-line	Intermediate	How-To	Workflow	No	Only implicitly (discusses branching/merging)	Yes	Mentioned (`merge`)	Basic

7 CONCLUSION

With learning resources, as in fashion, there is no such thing as ‘one size fits all’. The `git` version-control system has a lot of features and options, and can be employed in a wide variety of ways, so any complete explanation would be impractically long. This means that any given resource is likely to be making trade-offs between aspects such as the coverage of `git` features, the amount of prerequisite knowledge assumed, the expected workflow adopted, *etc.* More generally, the resources that we surveyed varied significantly in their style and quality.

Nonetheless, we can make a number of recommendations of good starting places for different sorts of learners, as summarised in the list that follows.

- The best tutorial for complete beginners to version control is “Version Control with Git” by Software Carpentry. However, this is with the caveat that it does not cover branches, so should be followed up by a resource that does; “Git Succinctly” for example. (Note that one of the authors is a certified Software Carpentry instructor, but also that that author was not responsible for determining the positive assessment provided here.)
- The best explanation of most common operations is provided within “Git Succinctly” by Ryan Hodson. “Ry’s Git Tutorial” (by the same author) covers the same material, plus a lot more as well, some of which is useful, but some is only of interest to very advanced users (*e.g.* explaining how to do a commit using only “plumbing”[52] commands.)
- The best explanation of underlying concepts for those who use `git` but don’t really understand it is “Git Happens” (a video of a conference talk) by Jessica Kerr.
- The best command-reference for everyday commands is “A Visual Git Reference” by Mark Lodato. It has some issues (like mentioning some potentially dangerous commands without explanation or warning), but is a good demonstration of (in particular) how `git` commands move data between the three, key storage areas (working directory, staging area, and commit history).
- The best overview of workflows, so as to help choose which to follow is “Comparing Git Workflows” (an episode of the “Coding Blocks” podcast) by Michael Outlaw, Allen Underwood, and Joe Zack. They describe a good range of approaches, are not dogmatic about which to use, and recognise the need to adapt any workflow to your particular situation/team.

As future work, we intend to engage with the authors of some of the above resources, or to directly participate in their collaborative development, taking into account what we have learned in performing the comparison between them.

Within the broader context of the New Zealand tertiary sector, *badges* to certify competence in specific technical skills are complementary to *micro-credentials* in the NZQA qualifications framework. While (NZQA) micro-credentials represent achievement across a coherent body of knowledge, (our) badges represent discrete units of competence. It may be that badges are used as part of micro-credentials, or even that a micro-credential could be created as collection of badges. For badges representing specific skills, however, we believe that many topics in computer science, like `git`, will be well supported by existing resources, and the task for the instructor is to identify which pathways through that material are most effective for their students. There is no single way to master skills-based topics like `git`, and it is not necessary to commit to a single version for all students.

[52]`git` distinguishes high-level “porcelain” commands for everyday use from low-level “plumbing” commands which are expected to be rarely invoked by experts.

REFERENCES

[1] Atlassian Corporation Plc. 2012. Merge or Rebase? http://blog.sourcetreeapp.com/2012/08/21/merge-or-rebase/

[2] Atlassian Corporation Plc. 2012. Smart branching with SourceTree and Git-flow. http://blog.sourcetreeapp.com/2012/08/01/smart-branching-with-sourcetree-and-git-flow/

[3] Stephen Ball. 2013. Deliberate Git. http://www.rakeroutes.com/blog/deliberate-git/ Slides available at http://speakerdeck.com/sdball/deliberate-git.

[4] Kerry Buckley. 2015. Git Masterclass. http://www.youtube.com/watch?v=L7_iMewv5vQ Slides available at http://speakerdeck.com/kerryb/git-masterclass.

[5] Buddy.Works. 2016. 5 types of Git workflow that will help you deliver better code. http://buddy.works/blog/5-types-of-git-workflows

[6] Cake Development Corporation. 2013. CakeDC Git Workflow. https://www.cakedc.com/git-workflow Also available as a pdf; slides from an explanatory talk at https://www.slideshare.net/lubomirstork9/cakedc-git-workflow-extension.

[7] Scott Chacon. 2011. GitHub Flow. http://scottchacon.com/2011/08/31/github-flow.html

[8] Spencer Christensen. 2014. Git Workflows That Work. https://www.endpoint.com/blog/2014/05/02/git-workflows-that-work

[9] Peter Cottle. 2019. Learn Git Branching. http://learngitbranching.js.org/

[10] Vincent Driessen. 2010. A successful Git branching model. http://nvie.com/posts/a-successful-git-branching-model/

[11] Roger Dudler. 2017. git - the simple guide. http://rogerdudler.github.io/git-guide/

[12] Nick Farina. 2011. Git Is Simpler Than You Think. http://nfarina.com/post/9868516270/git-is-simpler

[13] Courtenay Gasking. 2015. Version Control for Designers. http://web.archive.org/web/20150301060509/http://hoth.entp.com/output/git_for_designers.html Accessed via the internet archive.

[14] Marcus Geelnard. 2016. A stable mainline branching model for Git. https://www.bitsnbites.eu/a-stable-mainline-branching-model-for-git/

[15] Keith Gregory. 2014. Practical Git: A Workflow to Preserve Your Sanity. http://www.kdgregory.com/index.php?page=scm.git

[16] Ryan Hodson. 2012. *Ry's Git Tutorial*. RyPress, Boulder, CO. http://www.smashwords.com/books/view/498426 Website version still available on the internet archive http://web.archive.org/web/20161121145226/http://rypress.com:80/tutorials/git/index.

[17] Ryan Hodson. 2014. *Git Succinctly*. Syncfusion, Morrisville, NC. http://www.syncfusion.com/ebooks/git Also available as website http://code.tutsplus.com/series/git-succinctly--net-33581.

[18] Dirk Ifenthaler, Nicole Bellin-Mularski, and Dana-Kristen Mah (Eds.). 2016. *Foundation of Digital Badges and Micro-Credentials*. Springer.

[19] Joe James. 2014. Git Workflows for Pros: A Good Git Guide. http://www.toptal.com/git/git-workflows-for-pros-a-good-git-guide

[20] Jussi Judin. 2016. A succesful Git branching model considered harmful. http://barro.github.io/2016/02/a-succesful-git-branching-model-considered-harmful/

[21] Jessica Kerr. 2013. Git Happens. http://www.youtube.com/watch?v=Dv8I_kfrFWw

[22] Lars Kruse. 2014. An Automated Git Branching Flow. http://www.josra.org/blog/An-automated-git-branching-strategy.html

[23] Yosan Legaspi. 2017. 5 Git workflow best practices you've got to use. http://raygun.com/blog/git-workflow/ Updated Jan 2019.

[24] Sam Livingston-Gray. 2017. Think Like (a) Git. http://think-like-a-git.net/

[25] Mark Lodato. 2018. A Visual Git Reference. http://marklodato.github.io/visual-git-guide/index-en.html

[26] David Mahler. 2017. Core Concepts. http://www.youtube.com/user/mahler711/videos

[27] Nulab Inc. 2019. Git Tutorial. http://backlog.com/git-tutorial/

[28] Michael Outlaw, Allen Underwood, and Joe Zack. 2018. Comparing Git Workflows. http://www.codingblocks.net/podcast/comparing-git-workflows/

[29] Patrick Porto. 2018. 4 branching workflows for Git. http://medium.com/@patrickporto/4-branching-workflows-for-git-30d0aaee7bf

[30] Tom Preston-Werner. 2009. The Git Parable. http://tom.preston-werner.com/2009/05/19/the-git-parable.html

[31] Adam Ruka. 2015. GitFlow considered harmful. http://www.endoflineblog.com/gitflow-considered-harmful Follow-up at http://www.endoflineblog.com/follow-up-to-gitflow-considered-harmful.

[32] Adam Ruka. 2017. OneFlow - a Git branching model and workflow. https://www.endoflineblog.com/oneflow-a-git-branching-model-and-workflow

[33] Salesforce.com Inc. 2019. Git and GitHub Basics. http://trailhead.salesforce.com/en/content/learn/modules/git-and-git-hub-basics

[34] Benjamin Sandofsky. 2016. Understanding the Git Workflow. http://sandofsky.com/blog/git-workflow.html

[35] Corey Schafer. 2015. Git Tutorial for Beginners: Command-Line Fundamentals. http://www.youtube.com/watch?v=HVsySz-h9r4

[36] Corey Schafer. 2019. Git Tutorials. http://www.youtube.com/playlist?list=PL-osiE80TeTuRUfjRe54Eea17-YfnOOAx

[37] Isaac Schlueter. 2012. git-rebase. http://blog.izs.me/2012/12/git-rebase

[38] Michael Schwern. 2013. Git For Ages 4 And Up. http://www.youtube.com/watch?v=1ffBJ4sVUb4

[39] Paul Stadig. 2010. Thou Shalt Not Lie: git rebase, amend, squash, and other lies. http://paul.stadig.name/2010/12/thou-shalt-not-lie-git-rebase-ammend.html

[40] The Carpentries. 2019. Version Control with Git. http://swcarpentry.github.io/git-novice/

[41] Michelle Tilley and Katrina Uychaco. 2019. Visualizing Git. http://git-school.github.io/visualizing-git/

[42] Ryan Tomayko. 2008. The Thing About Git. http://tomayko.com/blog/2008/the-thing-about-git

[43] Linus Torvalds. 2009. Re: [git pull] drm-next. http://lwn.net/Articles/328438/

[44] Tutorials Point India Ltd. 2019. Git Tutorial. http://www.tutorialspoint.com/git/

[45] Gowtham Venkatesan. 2019. Learn the Basics of Git in Under 10 Minutes. http://www.freecodecamp.org/news/learn-the-basics-of-git-in-under-10-minutes-da548267cc91/

[46] Tommi Virtanen. 2019. Git for Computer Scientists. http://eagain.net/articles/git-for-computer-scientists/

[47] Wei Wang. 2017. Explain Git with D3. http://onlywei.github.io/explain-git-with-d3/

[48] Jim Weirich. 2018. Git Immersion. http://gitimmersion.com/

[49] Sandra Winkler. 2016. Git Workflow Explained — A Step-by-Step Guide. http://medium.com/@swinkler/git-workflow-explained-a-step-by-step-guide-83c1c9247f03