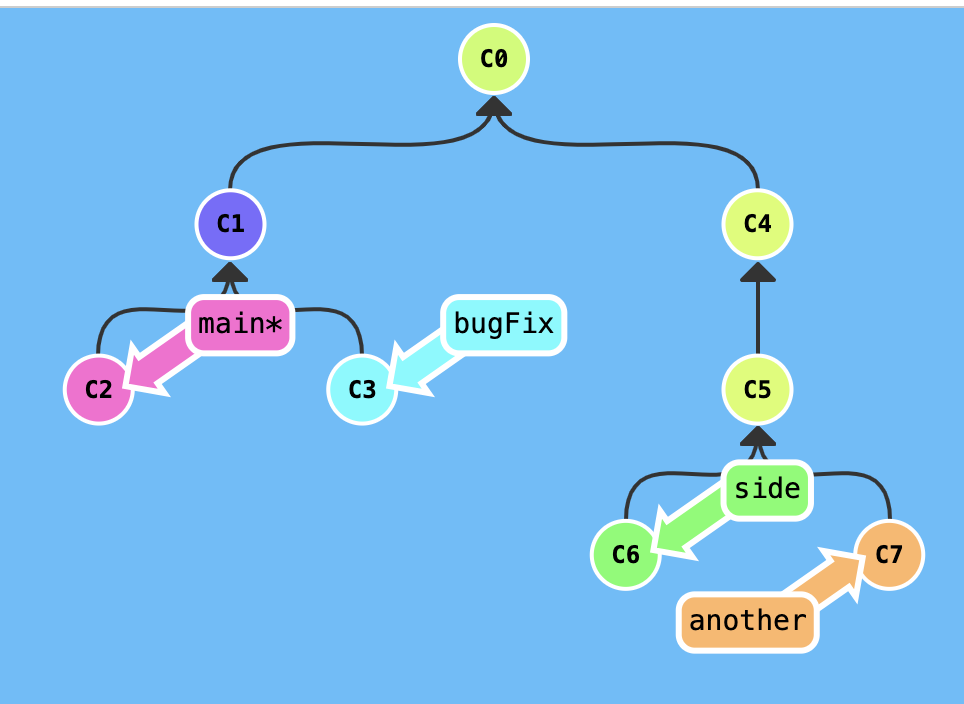
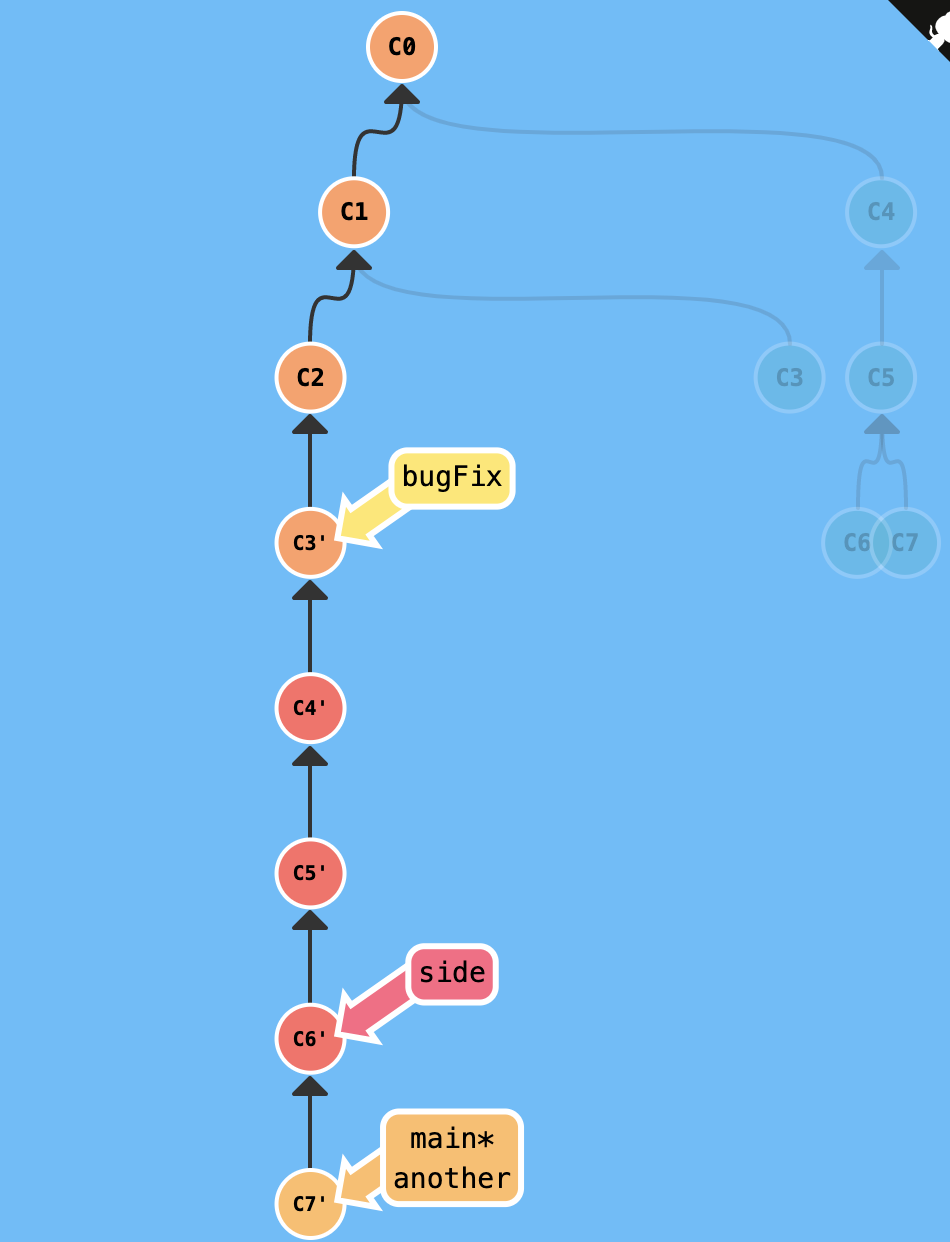
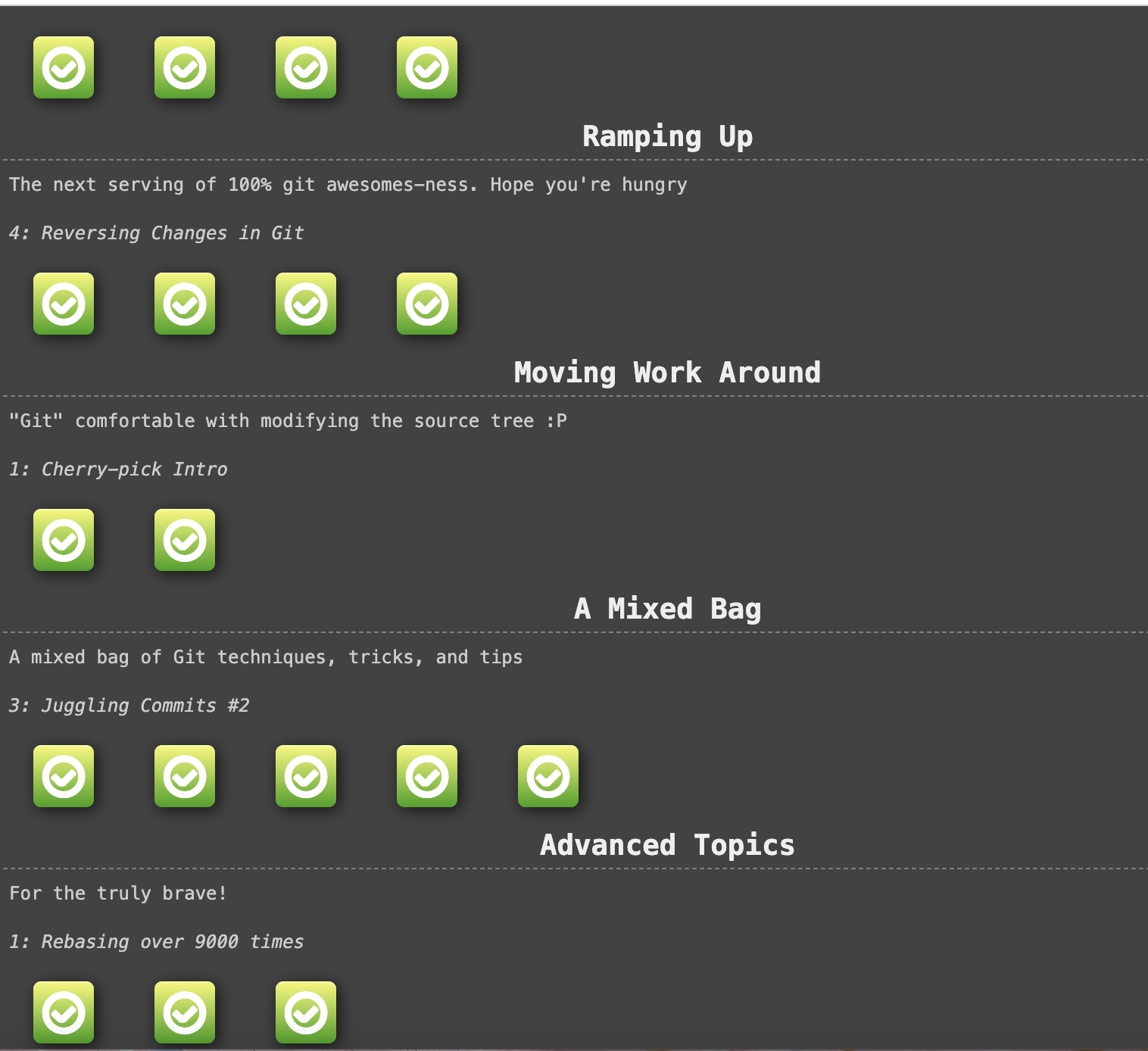
* Use ‘ show commands’ to see all commands in the terminal
* A commit in a git repository records a snapshot of all the (tracked) files in your directory.
  + Git also maintains a history of which commits were made when. That's why most commits have ancestor commits above them -- we designate this with arrows in our visualization.
  + ‘git commit’
* Branches are simply pointers to a specific commit -- nothing more.
  + Branch early and branch often.
  + To make a new branch named newImage do:
    - ‘git branch newImage’
    - Now the newImage refers to the commit of its ancestor ‘c1’.
* ‘git checkout’ means we want to commit with the branch as well as the main.
  + ‘git checkout newImage; git commit’
  + if you want to create a new branch AND check it out at the same time, you can simply type ‘git checkout -b [yourbranchname]’.
* Merging in Git creates a special commit that has two unique parents.
  + ‘git merge bugFix’ means to create a new branch with two parents, main and bugFix, but this is only the first step.
  + To merge main into bugFix we do:
    - ‘git checkout bugFix; git merge main’
* The second way of combining work between branches is rebasing. Rebasing essentially takes a set of commits, "copies" them, and plops them down somewhere else.
  + ‘git rebase main’
  + Then ‘git rebase bugFix’
* HEAD is the symbolic name for the currently checked out commit -- it's essentially what commit you're working on top of.
  + Head always points to the most recent commit
  + To detach the head, use the command ‘git checkout [name of head]’
* Must use ‘git log’ to see the hashes.
  + Moving upwards one commit at a time with ^
  + Moving upwards a number of times with ~<num>
  + Saying main^ is equivalent to the first parent of main.
  + main^^ is the grandparent (second ancestor) to main.
  + You can use ‘git checkout main^’ to detach the head to the first ancestor of main.
  + You can specify a number of commits using ~.
    - git checkout HEAD~4
      * This goes back to 4 ancestors ago.
  + You can directly reassign a branch to a commit with the -f option.
    - git branch -f main HEAD~3
      * moves (by force) the main branch to three parents behind HEAD.
* There are two primary ways to undo changes in Git -- one is using git reset and the other is using git revert.
* git reset
  + Reverses changes by moving a branch reference backwards in time to an older commit.
    - git reset HEAD~1 goes back to the last ancestor.
* git revert
  + In order to reverse changes and share those reversed changes with others, we need to use git revert.
    - git revert HEAD
* git cherry-pick
  + It's a very straightforward way of saying that you would like to copy a series of commits below your current location (HEAD).
  + git cherry-pick <Commit1> <Commit2> <...>
* Interactive Rebase
  + All interactive rebase means Git is using the rebase command with the -i option.
  + git rebase -i HEAD~4
* To copy only one commit over, do these commands:
  + git rebase -i
  + git cherry-pick
* You have some changes (newImage) and another set of changes (caption) that are related, so they are stacked on top of each other in your repository (aka one after another).
  + We will re-order the commits so the one we want to change is on top with git rebase -i
  + We will git commit --amend to make the slight modification
  + Then we will re-order the commits back to how they were previously with git rebase -i
  + Finally, we will move main to this updated part of the tree to finish the level (via the method of your choosing)
* git tag v1 c1
  + This tags C1 as v1
* Git Describe
  + Git describe can help you get your bearings after you've moved many commits backwards or forwards in history; this can happen after you've completed a git bisect (a debugging search)
  + git describe <ref>
    - <ref> is anything that can be committed
* Rebasing Multiple Branches
  + To merge branches do
    - git rebase main bugFix, this merges main and bug fix
    - So it merges bugfix under main.
    - So to solve this to the solution below we do
      * git rebase main bugfix
      * git rebase bugFix side
      * git rebase side another
      * git rebase another main



* git checkout main^ goes back to first parent (the left one) before main and separates the head.
  + git checkout main^2 goes back to the second parent
* To name a branch that is preexisting but has no name do
  + git branch name c2 (c2 being the position)
* git clone creates a clone of the repository
  + The new branch will be called o/main
  + When you commit the o/main, the o/main does not update, this is because o/.main will only update when the remote updates.
* git fetch
  + Fetches data from a remote repository.
  + Does not change anything about your local state
* git pull
  + Pulls data from other repository
* git fakeTeamwork
  + Plops down a commit on main but the remote was updated but we have not downloaded the commit yet because we havent done git fetch
* git push
  + Opposite of git pull
  + Uploads work
  + Updates the remote
  + git pull is just shorthand for a fetch and a merge.
  + git pull --rebase is shorthand for a fetch and a rebase.
* The remote rejected the push of commits directly to main because of the policy on main requiring pull requests to instead be used.