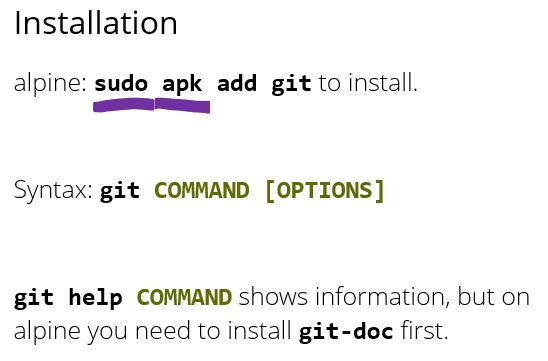
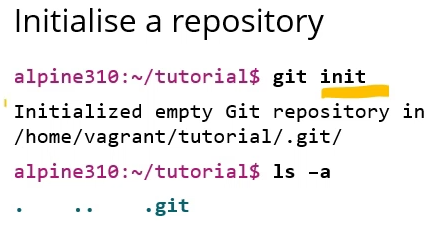
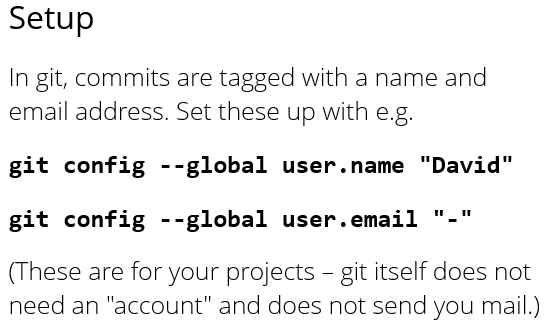
**Git 1**

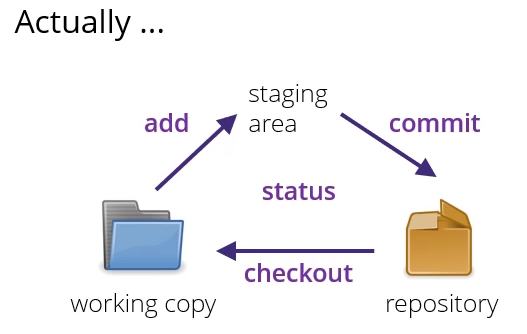
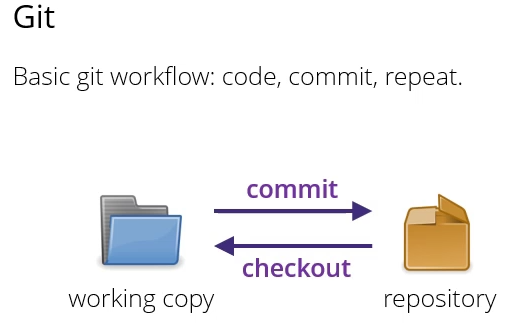
* Version control system to help you manage your source code
* Solves the problem of file system not doing versioning (complete history of your file), or issues with working together
* Basic principles of Git: work with code in a working folder, repository that stores complete history of project
* Most Linux distributions come with Git installed already, but not alpine as it is minimal



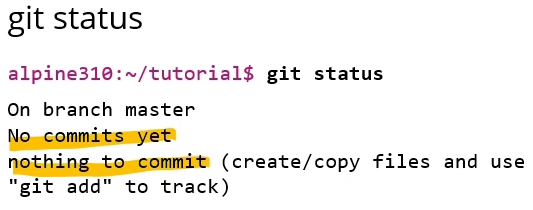
* Once you’ve installed Git, you need to configure it a bit

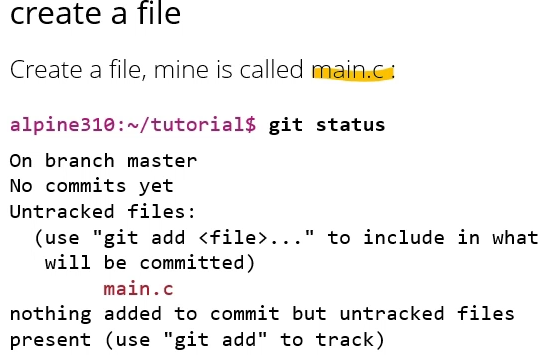


* A Git repository is just another folder, when you initialise it lives in a subfolder called .git

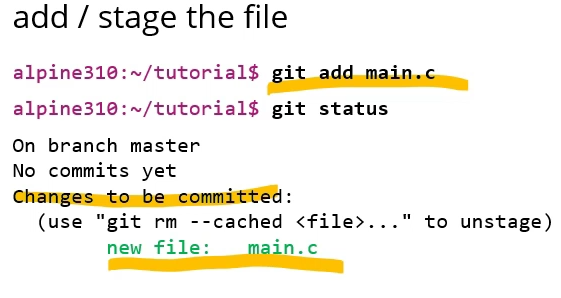


* If you only want to commit some files but not all, you can do that too
* Git status tells you what Git thinks is going on at the moment

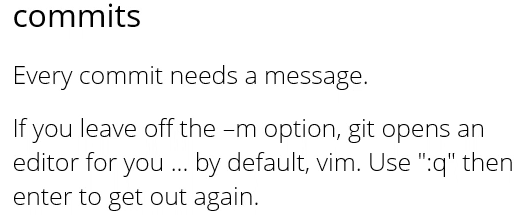
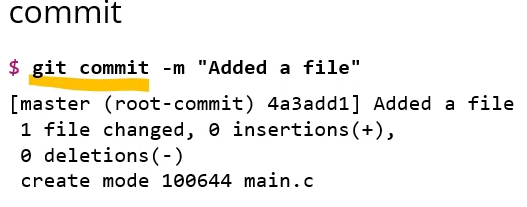




* Generally when Git shows something in red, it means you’ve done something you haven’t told Git about yet



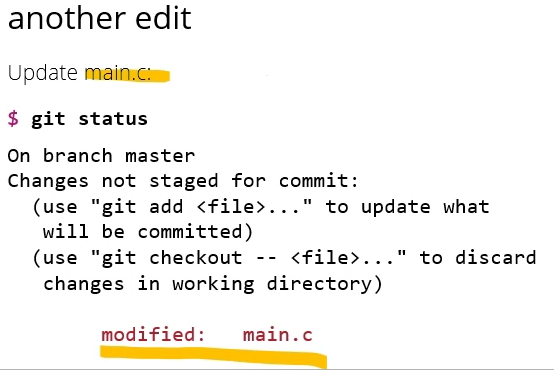
* Green indicates changes that Git knows about, and will be picked up in the next commit
* Every commit must have a message



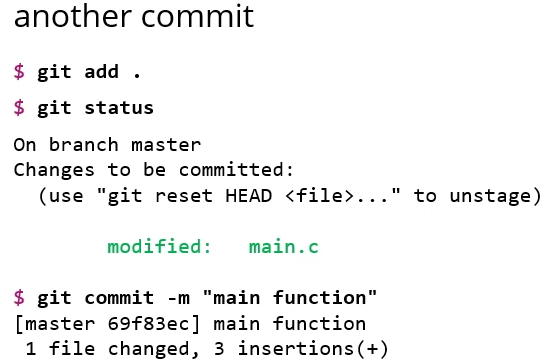
* Every commit gets an ID you can use to refer to a commit in a later command
* Generally should do lots of small commits not a few big ones – e.g. if you add 3 new features, should do at least 3 commits
* Use descriptive commit messages with a short first line (max 60-80 characters)
* For more detail in commit message:



* Can write a multi-line commit message using double quotes



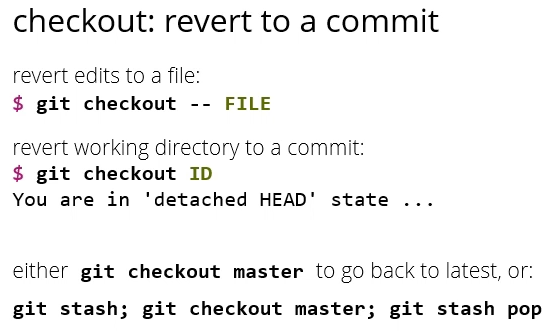
* The point of git making you first add and then commit your files is that you might not want to commit all your changes



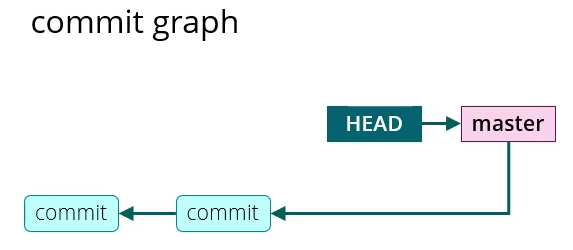
* You can add several files, but when you type git commit it will commit all files currently in the staging area
* Most of the time you just want to add what you added last time and can do this by git add .
  + . refers to the current directory
* If you want to look at all your commits so far, you can do this with git log



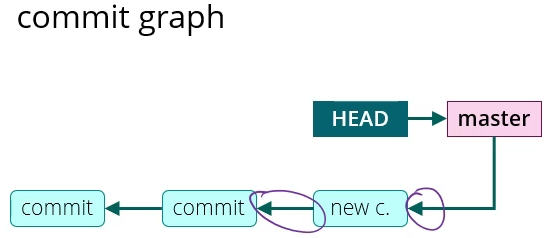
* To get stuff out of your repository into the working copy, we use the checkout command



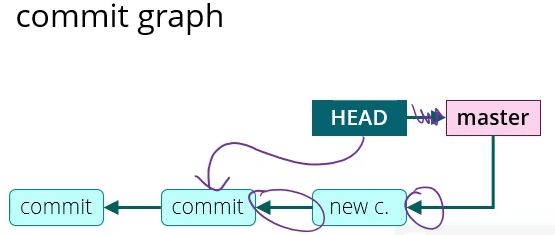
* Git checkout can undo edits
* To change whole WD to previous commit can use git checkout
  + If you’re in detached HEAD state you probably don’t want to make any further commits because they won’t go anywhere
* Internally, a commit is a data structure that stores a few things like a commit message, the name and email of the committer, the changes applied to the commit



* A commit also contains a list of pointers, so these can be used to build linked lists
* Pointers are called parents and point to previous commits
* Separately to the commits, there are several data structures that just contain pointers
* One of these is called branches, or refs, of which master is one
* Head generally points at the state of what is currently in your working copy (here master)
* When you make a new commit, git will create a new data structure and then adjust the pointers
* It will take the head structure, follow that along until it finds a commit and then relink the pointers



* If you check out an earlier commit, git will take its head pointer and point that at the commit you checked out



* So detached head state means that head is not pointing at a branch anymore, its pointing directly at a commit (why you don’t want to make any new commits)
* Checkout master points head back to master again and restores the files that master was pointing at
* Summary:

