Choose https, git clone URL:

Download a local copy of repo

‘cd’ shows current directory. cd with some other name moves to that location

ls lists all files in it. -a shows all, even hidden files, -al is preferred, shows long format of files

Git status shows branch status wrt master

Each commit has a unique identifier determined by sha-1

**Structure:**

**Local Remote**

| **Working Staging Repository**  **directory area (.git folder)** |  |
| --- | --- |

Every commit done locally puts us ahead of the remote master branch

Git commit -m ‘rthttyh grg’, or git commit, then type name

Git pull origin master pulls down our remote repo to local to ensure we have the up-to-date local version of the remote repo.

Best practice is always to do a pull before a push. (push is like commit, but to the remote version)

Git push origin master

Notepad ~/.gitconfig

Note that notepad command can be replaced with any text editor for editing the file(which has email id and password)

Git commit -am - used to both add and commit to stage in one step (not best practice, but usually okay for one or two files being changed, also if they are already being tracked by git)

a tracked file is any file that Git is aware of and tracking actively.

Any file added to index, stage or committed to git repo

Git ls-files - shows all tracked files

Even when editing or deleting a file, we must first stage the changes.

Therefore ‘git add <filename> or add . ‘ first before committing.

Type git status to see what has been staged and what has not etc.

Mkdir with -p helps create a directory structure.

Add all files/all files recursively, use git add .

Note: most git commands work anywhere in the git repository folders

Git mv a.txt b.txt, renames file from a.text to b.text in git

mv a.txt b.txt, renames file from a.text to b.text in os level

Mv filename .. - moves file one level down in the directory

Untracked files, simply rm <filename>

Tracked files, have to commit the deletion,’ git rm filename’ and ‘git commit -m ‘dyreu’’

To bring back a deleted file, git reset head won’t do it. It will only unstage the deletion, but in your directory, file doesn’t appear until you do git checkout

Git add -A, used to add git renames and deletions

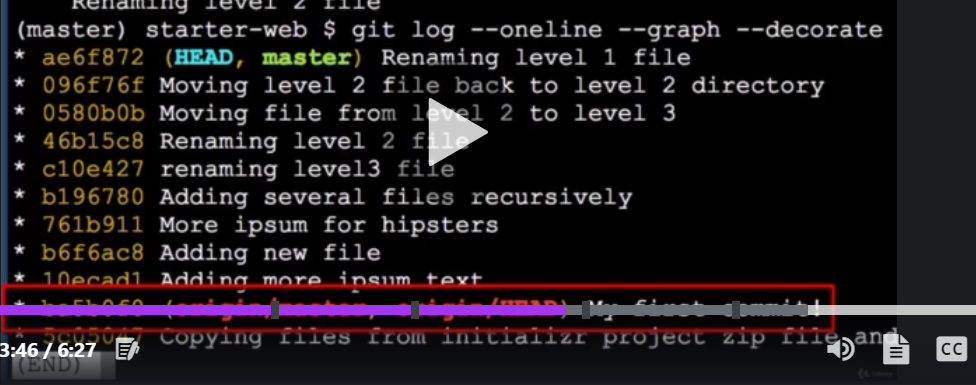
Then git commit

Rm -rf <folder> removes folder and all subfolders and files.(Caution! Powerful command)

Then git add -A then git commit

‘git log’ shows commit history

Commit ids are long, can abbreviate them to 6-7 characters to uniquely identify



Git log - shows commit history

"git log --since="3 days ago"

Shows commits of last 3 days

Git log --filename shows commit history of the file



Git show commit id shows info on that particular id.

Git aliases:

To make commands shorter

Git congif --global(available anywhere in repo) alias.<aliasname>

Notepad .~/gitconfig, put alias in the file as

[alias]

<filename = log --all --decorate(any command you want to make shorter)

Git <aliasname>

->

.gitignore file:

Notepad .gitignore creates the file

Git add .gitignore

Git commit -m ‘eghrjrt’

->

Sync local and remote repo, remote is to be pulled before changes are pushed

Git diff - shows differences(if any) between what is in local and what has been staged. It shows the lines that were changed. If no changes, it shows nothing

‘Git diff head’ shows difference between working directory and last commit

Git difftool --staged HEAD shows git repo at head and our staging area (uses p4merge for this)

Git diff -- <filename>, shows only for that file

Git diff commit-id1 commit-id2

Git diff commit-id1 HEAD

Git diff HEAD HEAD^ (b/w head and head-1 commit

During git diff, if you see a /dev/null in commit history / comparisons means that file didnt exist in that commit

Git diff master origin/master (finds diff between local repo and remote repo

Note: branch merge conflicts occured

Branches & Merging

Git branch -a will list all branches in local and in the repo. It also shows which branch we’re on.

Git branch -m oldname newname (renames branches)

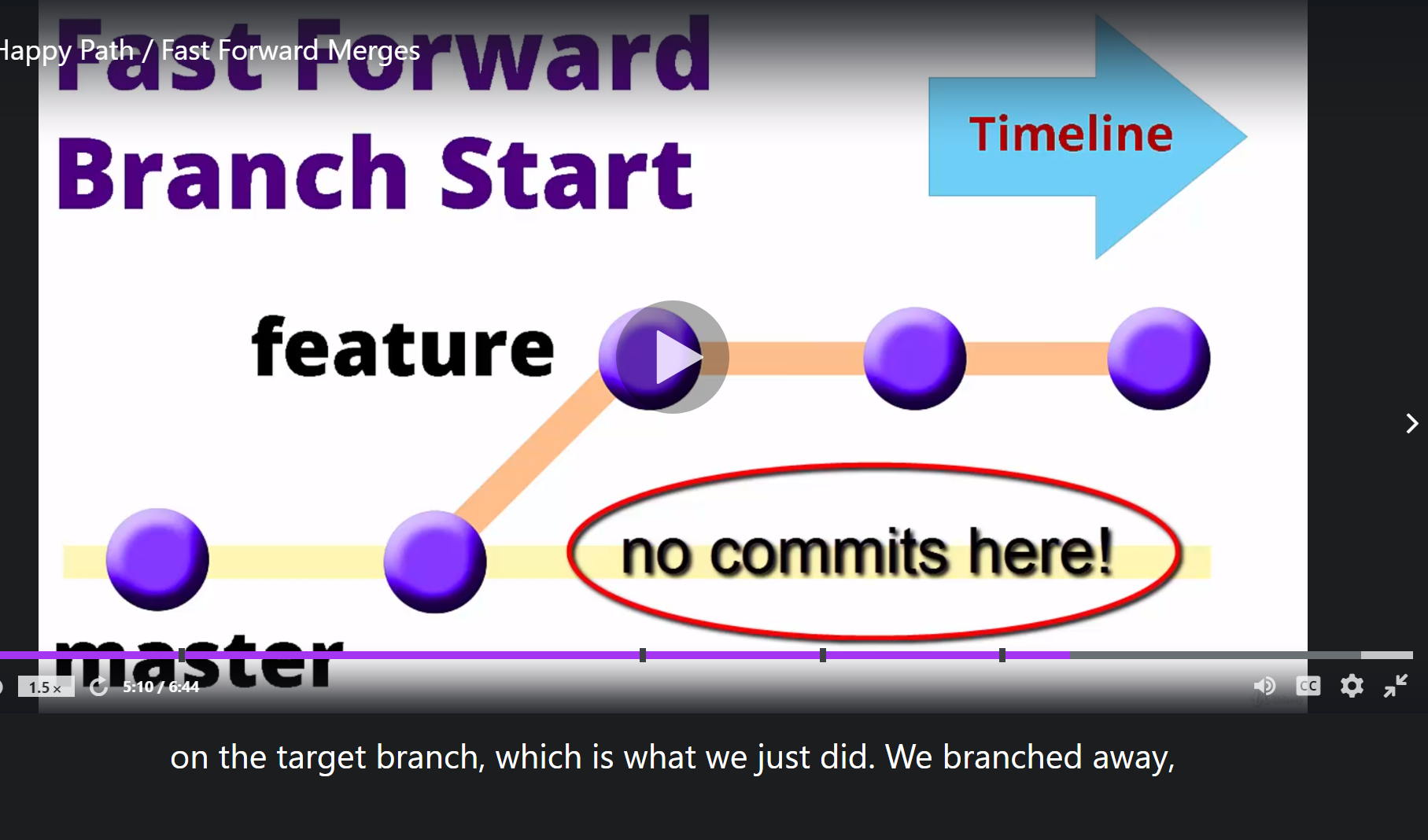
Git branch name (creates)

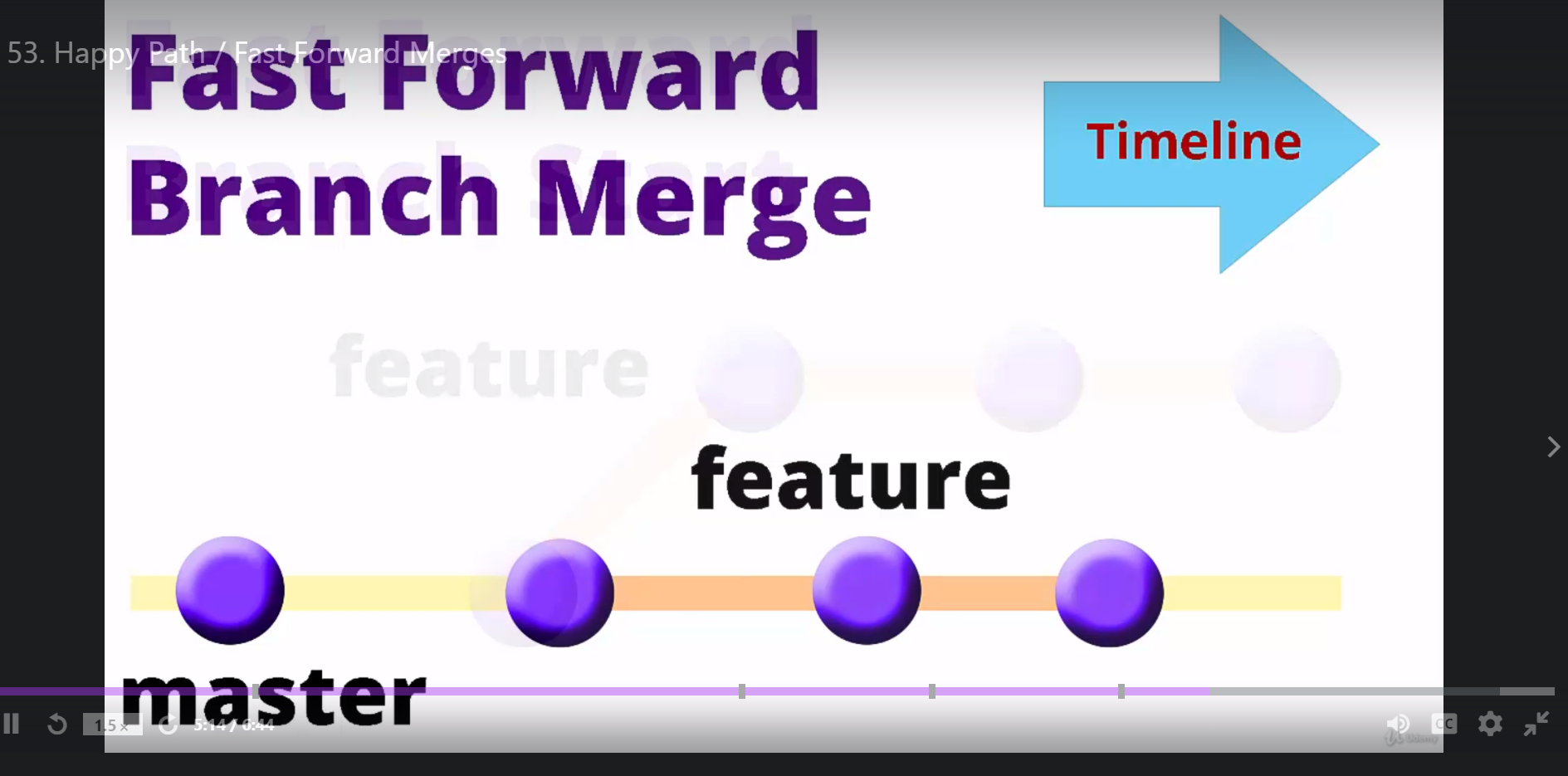
Git checkout name(switches to that branch)

Git branch -d name (deletes branch)(shouldnt be on this branch while deleting though, wont work)

Or

Git checkout -b newbranchname (creates and switches to that branch)



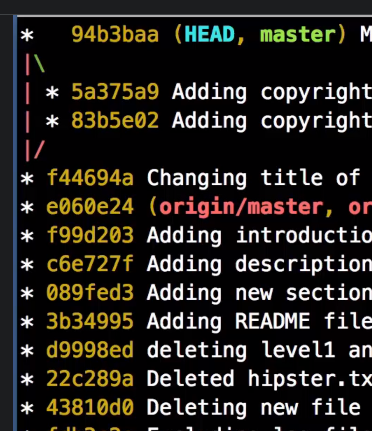


Think of branches as only pointers, nothing too complex. Deleting branch only deletes pointer.

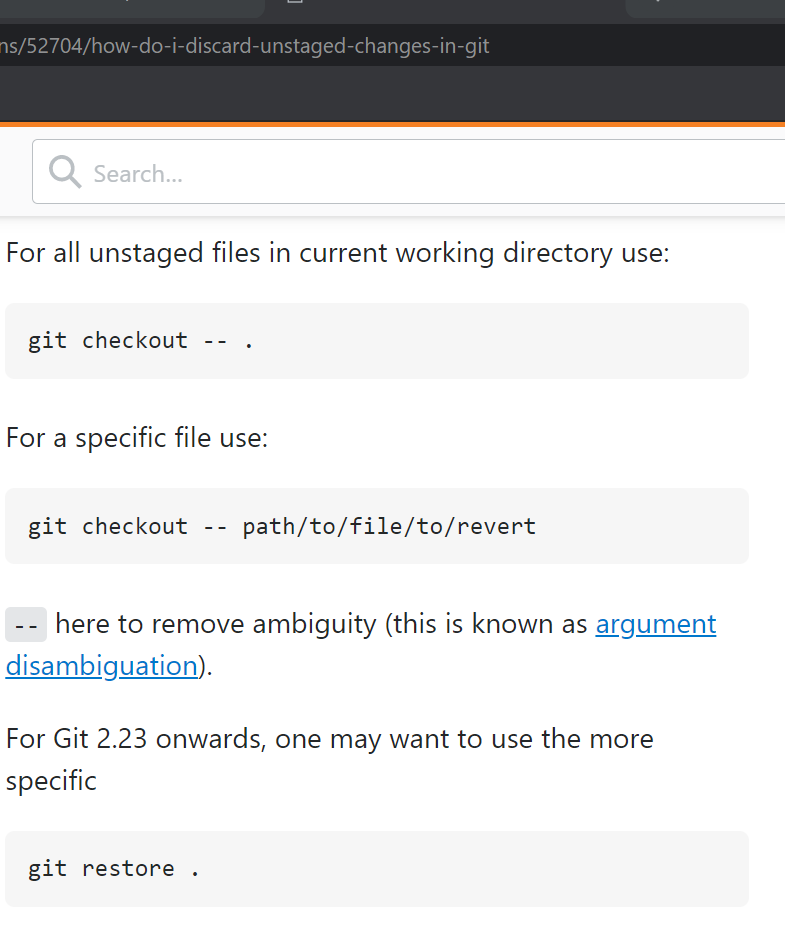
origin/master is the repo master branch, and master is local master, head is latest commit.

No fast-forward(used to preserve the fact that we branched off):

That is , it can show us that we branched off, but doesnt have a label associated with that branch anymore after we delete it.



Delete unstaged changes



Automatic changes occur when we create a branch, make commits on it, move back to master and click merge

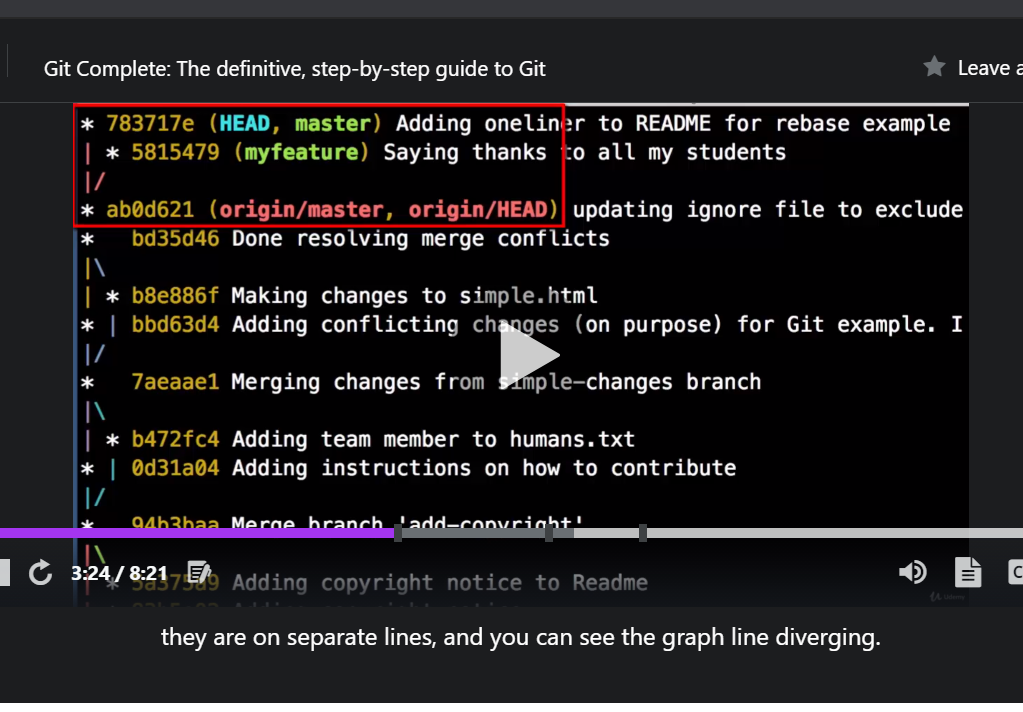
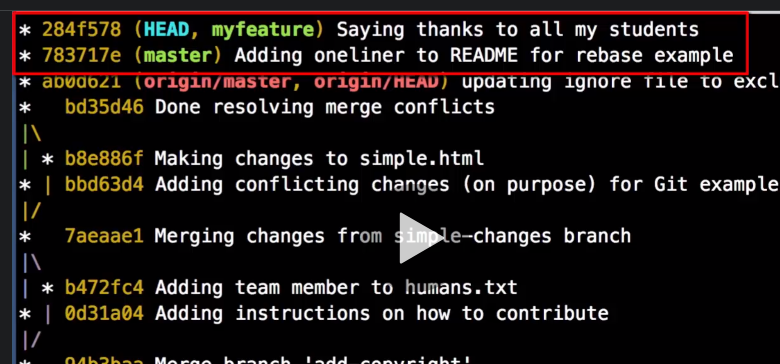
git log --oneline --graph --decorate --all

Is used to visualise the branches, can help us show if branches have merged or diverged etc as well

If a merging conflict has occurred and auto-merge fails, then we are in a neither clean nor staged area, sort of in-between state.

Use git mergetool (that uses p4merge) and use it to resolve & merge the conflicts. Then commit the changes. So branch changes from eg: master|merging to just master. This creates a list of orig files to change any of these conflict ‘decisions’. Make sure to add these merge temp files to git ignore

**Rebase**

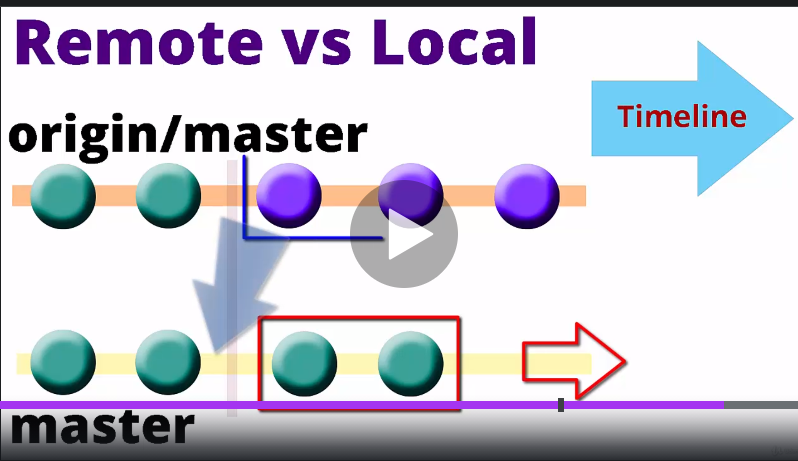
**  
**

Basically we replay/undo the feature change, do the change in master, then redo the change in feature, so that it incorporates any changes that have happened in master

To resolve merge conflict from rebase, we can use p4merge

If you think that master may have changed while you’re on a branch.then instead of git pull, do git fetch(which is a non-destructive command), it only updates reference so that when you do git status, it can tell you master is no longer the same

‘Git pull **--rebase** origin master‘ accommodates those changes in master to our local master, then we can continue and commit and push our changes to repo master like we usually do for local changes.





**Stash** (note: git stash only works on tracked files)

‘Git stash’ puts away unfinished work in order to let us do another/more urgent commit. After that commit is done, then we can do ‘git stash apply’ to bring those changes back and finish the work, then commit it. ‘Git stash list’ shows which contents are available in stash. ‘Git stash drop’ deletes those contents saved in stash as we have already applied it to our file and have no need of it.

‘Git stash -u’ includes **untracked files** as well, and brings them back.

Note: instead of ‘git stash apply’ and then ‘git stash drop’, we can use one command ‘git stash pop’.

Might use **multiple** stashes? Then use ‘**git stash save “name of stash”** ‘.

Git stash list shows in the reverse order of stashes - newest one at 0, oldest at 2.

stash@{0} : latest stash

stash@{1} : 2nd latest stash

stash@{2} : oldest stash

Git stash show stash@{1} shows the changes in the stash at 1.

Reapply a specific stash using ‘git stash apply stash@{1}’

Git stash drop stash@{1} deletes that stash alone

Git stash clear, deletes all stashes

Stash from one branch, move to another:

Eg: if you made some changes on master, but don’t want to commit there, then ‘git stash -u’ (even untracked files) there, and then do ‘git stash branch newbranch (it creates and puts changes in that branch, ie, applies stash.

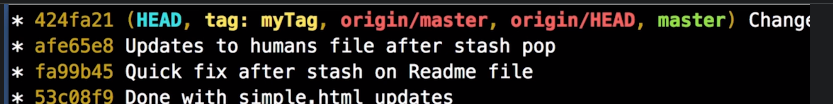
Tagging :

As we make lots of changes in our repo, we need a way to mark significant events in our git repo, and that is possible through tagging. Tags are labels that can be applied to any commit in history.

‘Git tag tagname’ creates tag. ‘Git tag --list’ shows tag list(careful about double dashes, otherwise it will just create a tag called list)

Other git commands can use commit id or tag alrenatively, eg: git show tagname

Git tag --delete tagname - deletes tag



Git tag -a tagname (annotated tags) (tag with a message, eg: tag name will have version name ‘v 1.0’ and annotation tells us what the release contained

This command takes use to the tag message page, where we can add relevant info. First line in that tag file/message becomes commit message name, so no need to add message to those commits

Git diff tagname1 tagname2

Bring up first different file, compare changes and press ctrl+q and then it moves to next file

To apply tag to a specific commit, git tag -a tagname commit-id

Update a tag:

1. Delete the tag assigned to wrong commit and reassign to right tag

or

1. Git tag -a <wronglyassignedtagname> -f <correct-commit-id>. It automatically lets us edit tag message

‘Git Push origin tagname’ pushes that release/tag(with all changes and notes, to github

In github, tags are tracked, with the three dots next to it storing the message/annotation.

To delete a tag from repo master, use ‘git push origin :tagname’. So it may be on local, but not the remote repo.

**Reset and reflog**

Git reset HEAD^1 (or) git reset HEAD^ (or) git reset HEAD^^ : goes back/up by one commit

Git log - shows history. head^2 goes back 2 levels/commits

Git reflog - shows us the log of each commit(even after we have ‘removed’ using reset. Note, this can be done upto 60 days, so careful about getting the commit ids so removed before this timeframe.

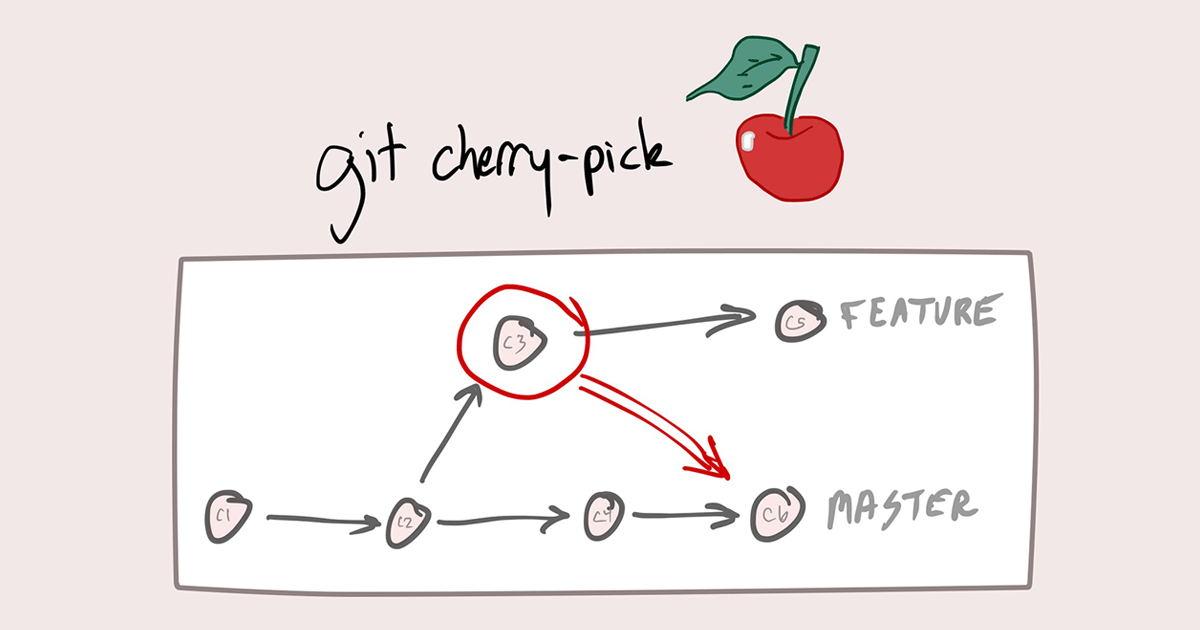
Git reset <commit-id> removes all commits till that id, it can also be used to move forward to one of the commit id’s that were removed earlier in (git reset). Git reflog shows back everything

**To compare two branches ( via UI/directly ):** In git, can create a new branch and then create a pull-request on it to compare changes. Don’t really have to follow up on it, can just use it to see differences b/w branch and master

**To compare two branches ( via CLI ):** switch to branch using git checkout ‘git checkout branchname2’, ‘git diff master branchname2’. Then switch back to master and delete old branch.

It is important to have a clean working directory in git.use ‘git status’ to check

**Cherry-pick:**

****

Git cherry-pick c3