

<https://www.freecodecamp.org/news/git-cheat-sheet/>

Git is a distributed version control system that helps developers collaborate on projects of any scale.

Linus Torvalds, the developer of the Linux kernel, created Git in 2005 to help control the Linux kernel's development.

**What is a Distributed Version Control System?**

A distributed version control system is a system that helps you keep track of changes you've made to files in your project.

This change history lives on your local machine and lets you revert to a previous version of your project with ease in case something goes wrong.

Git makes collaboration easy. Everyone on the team can keep a full backup of the repositories they're working on on their local machine. Then, thanks to an external server like BitBucket, GitHub or GitLab, they can safely store the repository in a single place.

This way, different members of the team can copy it locally and everyone has a clear overview of all changes made by the whole team.

Git has many different commands you can use. And I've found that these fifty are the ones I use the most often (and are therefore the most helpful to remember).

So I have written them down and thought it'd be nice to share them with the community. I hope you find them useful – Enjoy.

**How to check your Git configuration:**

The command below returns a list of information about your git configuration including user name and email:

git config -l

**How to setup your Git username:**

With the command below you can configure your user name:

git config --global user.name "Fabio"

**How to setup your Git user email:**

This command lets you setup the user email address you'll use in your commits.

git config --global user.email "signups@fabiopacifici.com"

**How to cache your login credentials in Git:**

You can store login credentials in the cache so you don't have to type them in each time. Just use this command:

git config --global credential.helper cache

**How to initialize a Git repo:**

Everything starts from here. The first step is to initialize a new Git repo locally in your project root. You can do so with the command below:

git init

**How to add a file to the staging area in Git:**

The command below will add a file to the staging area. Just replace filename\_here with the name of the file you want to add to the staging area.

git add filename\_here

**How to add all files in the staging area in Git**

If you want to add all files in your project to the staging area, you can use a wildcard . and every file will be added for you.

git add .

**How to add only certain files to the staging area in Git**

With the asterisk in the command below, you can add all files starting with 'fil' in the staging area.

git add fil\*

**How to check a repository's status in Git:**

This command will show the status of the current repository including staged, unstaged, and untracked files.

git status

**How to commit changes in the editor in Git:**

This command will open a text editor in the terminal where you can write a full commit message.

A commit message is made up of a short summary of changes, an empty line, and a full description of the changes after it.

git commit

**How to commit changes with a message in Git:**

You can add a commit message without opening the editor. This command lets you only specify a short summary for your commit message.

git commit -m "your commit message here"

**How to commit changes (and skip the staging area) in Git:**

You can add and commit tracked files with a single command by using the -a and -m options.

git commit -a -m "your commit message here"

git commit -am "your commit message here"

but this cannot be used for new files that have never been added to the tracked files – i.e. only for modified files which are by nature already tracked.

**How to see your commit history in Git:**

This command shows the commit history for the current repository:

git log

**How to see your commit history including changes in Git:**

This command shows the commit's history including all files and their changes:

git log -p

**How to see a specific commit in Git:**

This command shows a specific commit.

Replace commit-id with the id of the commit that you find in the commit log after the word commit.

git show commit-id

**How to see log stats in Git:**

This command will cause the Git log to show some statistics about the changes in each commit, including line(s) changed and file names.

git log --stat

**How to see changes made before committing them using "diff" in Git:**

You can pass a file as a parameter to only see changes on a specific file.  
git diff shows only unstaged changes by default.

We can call diff with the --staged flag to see any staged changes.

git diff

git diff all\_checks.py

git diff --staged

**How to see changes using "git add -p":**

This command opens a prompt and asks if you want to stage changes or not, and includes other options.

git add -p

**How to remove tracked files from the current working tree in Git:**

This command expects a commit message to explain why the file was deleted.

git rm filename

**How to rename files in Git:**

This command stages the changes, then it expects a commit message.

git mv oldfile newfile

**How to ignore files in Git:**

Create a .gitignore file and commit it.

**How to revert unstaged changes in Git:**

git checkout filename

**How to revert staged changes in Git:**

You can use the -p option flag to specify the changes you want to reset.

git reset HEAD filename

git reset HEAD -p

**How to amend the most recent commit in Git:**

git commit --amend allows you to modify and add changes to the most recent commit.

git commit --amend

!!Note!!: fixing up a local commit with amend is great and you can push it to a shared repository after you've fixed it. But you should avoid amending commits that have already been made public.

**How to rollback the last commit in Git:**

git revert will create a new commit that is the opposite of everything in the given commit.  
We can revert the latest commit by using the head alias like this:

git revert HEAD

**How to rollback an old commit in Git:**

You can revert an old commit using its commit id. This opens the editor so you can add a commit message.

git revert comit\_id\_here

**How to create a new branch in Git:**

By default, you have one branch, the main branch. With this command, you can create a new branch. Git won't switch to it automatically – you will need to do it manually with the next command.

git branch branch\_name

**How to switch to a newly created branch in Git:**

When you want to use a different or a newly created branch you can use this command:

git checkout branch\_name

git switch branch\_name

**How to list branches in Git:**

You can view all created branches using the git branch command. It will show a list of all branches and mark the current branch with an asterisk and highlight it in green.

git branch

**How to create a branch in Git and switch to it immediately:**

In a single command, you can create and switch to a new branch right away.

git checkout -b branch\_name

is the same as git switch -c branch\_name

**How to delete a branch in Git:**

When you are done working with a branch and have merged it, you can delete it using the command below:

git branch -d branch\_name

**How to merge two branches in Git:**

To merge the history of the branch you are currently in with the branch\_name, you will need to use the command below:

git merge branch\_name

To merge branch a into branch b, fist switch to the target branch

(

git switch b (target branch)

git merge a (source branch)

)

i.e. merge feature (back) into main:   
git switch main

git merge feature

**How to show the commit log as a graph in Git:**

We can use --graph to get the commit log to show as a graph. Also,  
--oneline will limit commit messages to a single line.

git log --graph --oneline

**How to show the commit log as a graph of all branches in Git:**

Does the same as the command above, but for all branches.

git log --graph --oneline --all

**How to abort a conflicting merge in Git:**

If you want to throw a merge away and start over, you can run the following command:

git merge --abort

**How to add a remote repository in Git**

This command adds a remote repository to your local repository (just replace https://repo\_here with your remote repo URL).

git add remote https://repo\_here

**How to see remote URLs in Git:**

You can see all remote repositories for your local repository with this command:

git remote -v

**How to get more info about a remote repo in Git:**

Just replace origin with the name of the remote obtained by  
running the git remote -v command.

git remote show origin

**How to push changes to a remote repo in Git:**

When all your work is ready to be saved on a remote repository, you can push all changes using the command below:

git push

**How to pull changes from a remote repo in Git:**

If other team members are working on your repository, you can retrieve the latest changes made to the remote repository with the command below:

git pull

**How to check remote branches that Git is tracking:**

This command shows the name of all remote branches that Git is tracking for the current repository:

git branch -r

**How to fetch remote repo changes in Git:**

This command will download the changes from a remote repo but will not perform a merge on your local branch (as git pull does that instead).

git fetch

**How to check the current commits log of a remote repo in Git**

Commit after commit, Git builds up a log. You can find out the remote repository log by using this command:

git log origin/main

**How to merge a remote repo with your local repo in Git:**

If the remote repository has changes you want to merge with your local, then this command will do that for you:

git merge origin/main

**How to get the contents of remote branches in Git without automatically merging:**

This lets you update the remote without merging any content into the  
local branches. You can call git merge or git checkout to do the merge.

git remote update

**How to push a new branch to a remote repo in Git:**

If you want to push a branch to a remote repository you can use the command below. Just remember to add -u to create the branch upstream:

git push -u origin branch\_name

**How to remove a remote branch in Git:**

If you no longer need a remote branch you can remove it using the command below:

git push --delete origin branch\_name\_here

**How to use Git rebase:**

You can transfer completed work from one branch to another using git rebase.

git rebase branch\_name\_here

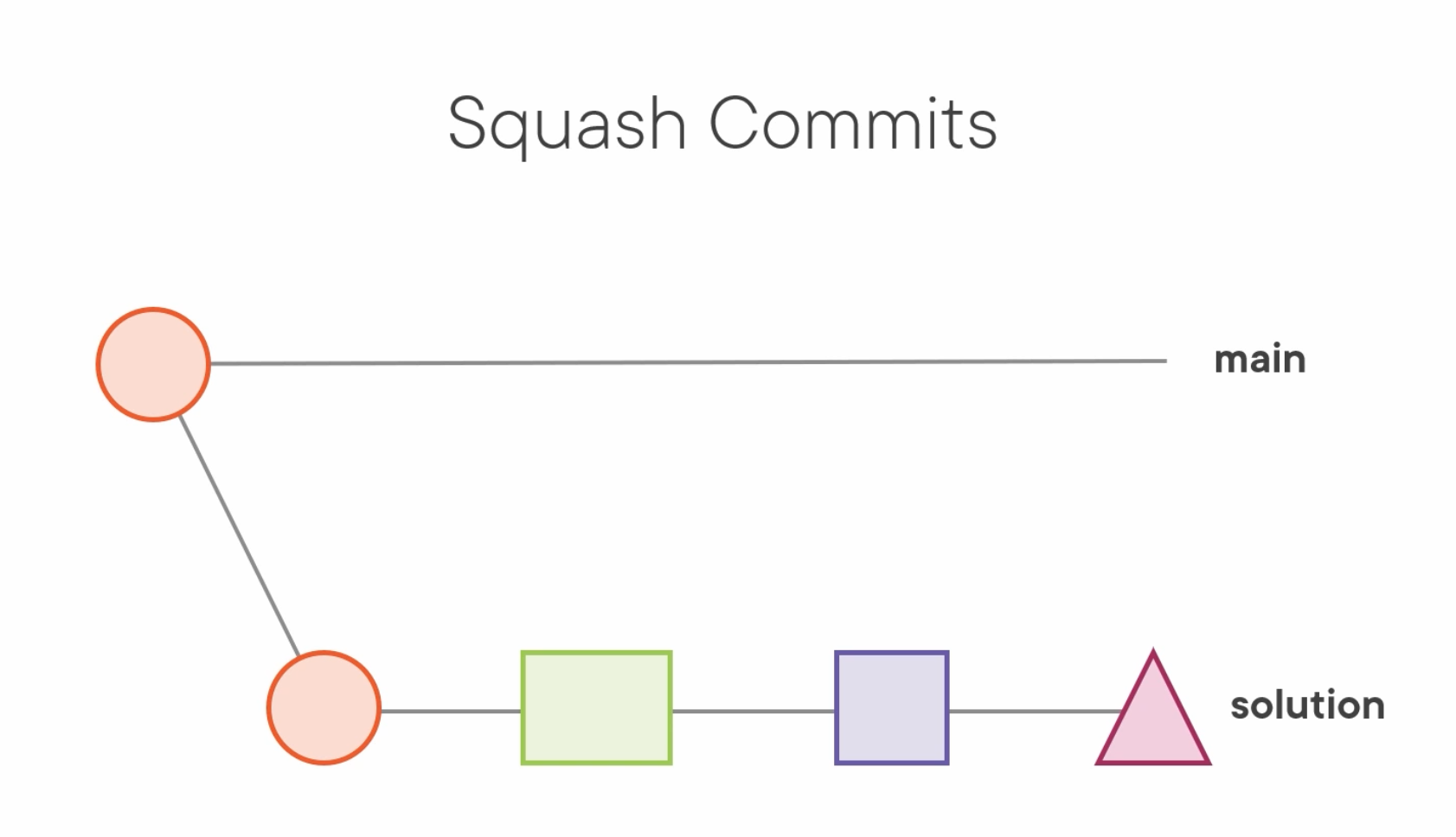
Git Rebase can get really messy if you don't do it properly. Before using this command I suggest that you re-read the official documentation [here](https://git-scm.com/book/it/v2/Git-Branching-Rebasing)

Do NOT rebase when working on a public branch!!

Be sure to check the team guidelines about their use of rebase – some teams decide not to use rebase at all.

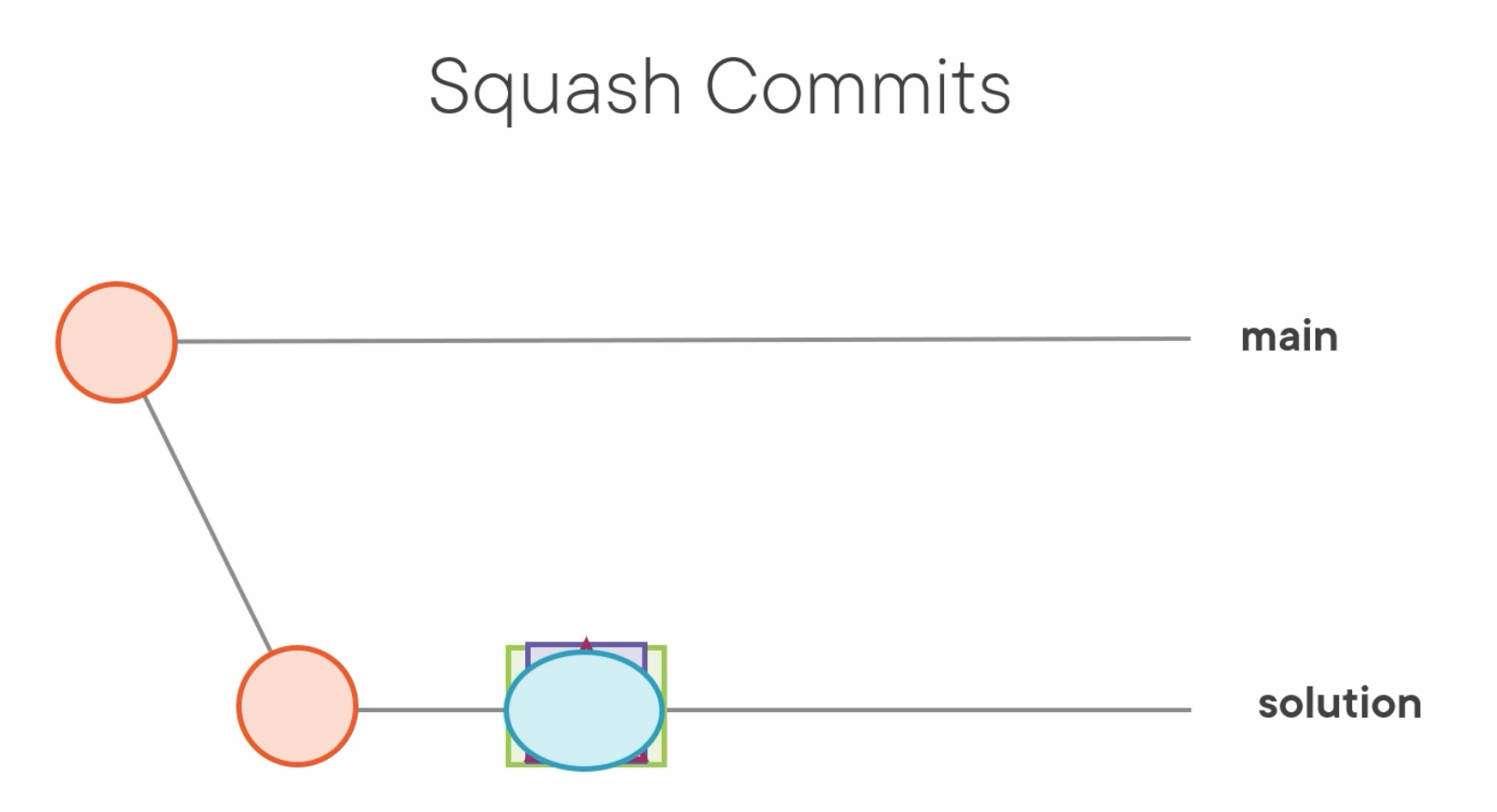
**Rebase scenarios:**

**Clean up history**Clean up your local history before sharing a branch



Squashing multiple commits into one commit

git rebase -i <commit-sha (of the green commit above)>



**Pull without merge**Pull changes from a branch into your branch without performing a merge.

**How to run rebase interactively in Git:**

You can run git rebase interactively using the -i flag.  
It will open the editor and present a set of commands you can use.

git rebase -i master

# p, pick = use commit

# r, reword = use commit, but edit the commit message

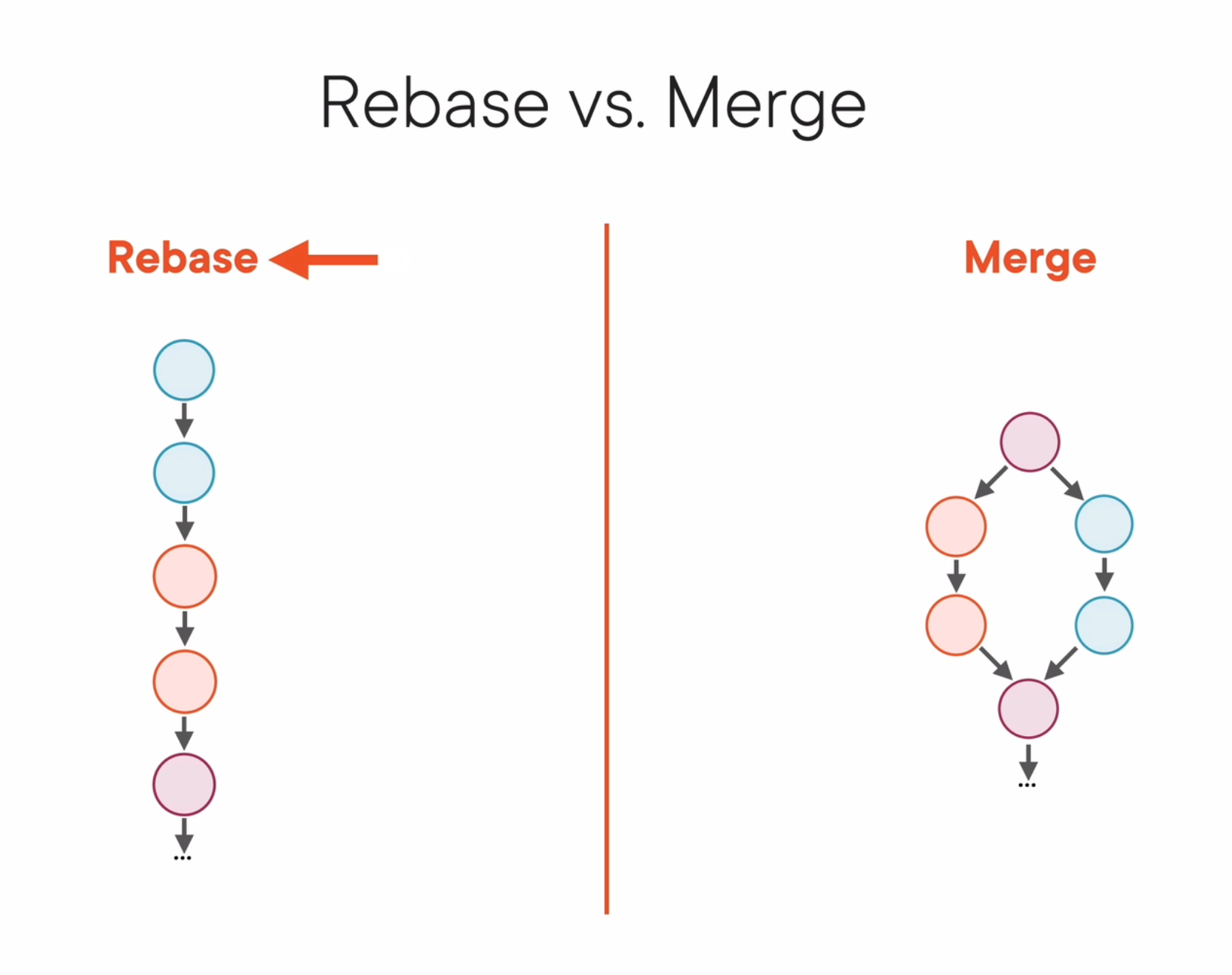
# e, edit = use commit, but stop for amending

# s, squash = use commit, but meld into previous commit

# f, fixup = like "squash", but discard this commit's log message

# x, exec = run command (the rest of the line) using shell

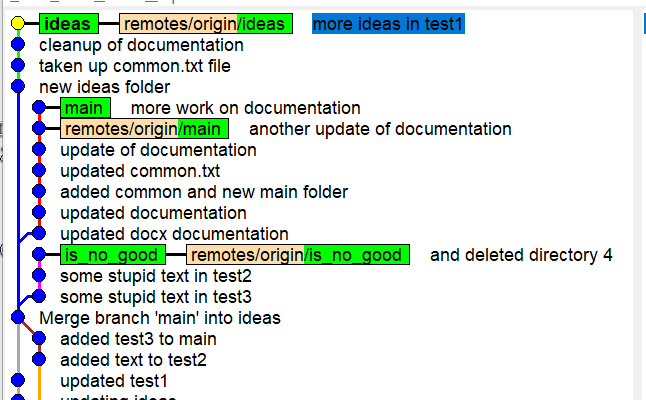
# d, drop = remove commit



e.g. squash commits

starting situation:

|  |
| --- |
| PeterS@PCW10 MINGW64 /e/PeterS/t3 (ideas)  $ git log --oneline  903ee39 (HEAD -> ideas, origin/ideas) more ideas in test1  26466a8 cleanup of documentation  4b52fe5 taken up common.txt file  fb650d4 new ideas folder  3d3f585 Merge branch 'main' into ideas  42517a3 added test3 to main  4ab5490 updated test1  b84328a updating ideas  7ce1402 added text to test2  d0b4650 extended cheat sheet  f4c8867 new desktop.ini  fabe092 first commit |



|  |
| --- |
| PeterS@PCW10 MINGW64 /e/PeterS/t3 (ideas)  $ git merge-base ideas main  3d3f5852b4a976a9cc889eaaa3cf7ea85fb7a4dd |

git rebase -i 3d3f585

git will present the file below with each commit that is part of the rebase. Instructions need to be given in the first lines, but possibilities are shown in the commented lines below – note also the very last line…

|  |
| --- |
| pick fb650d4 new ideas folder  pick 4b52fe5 taken up common.txt file  pick 26466a8 cleanup of documentation  pick 903ee39 more ideas in test1  # Rebase 3d3f585..903ee39 onto 3d3f585 (4 commands)  #  # Commands:  # p, pick <commit> = use commit  # r, reword <commit> = use commit, but edit the commit message  # e, edit <commit> = use commit, but stop for amending  # s, squash <commit> = use commit, but meld into previous commit  # f, fixup [-C | -c] <commit> = like "squash" but keep only the previous  # commit's log message, unless -C is used, in which case  # keep only this commit's message; -c is same as -C but  # opens the editor  # x, exec <command> = run command (the rest of the line) using shell  # b, break = stop here (continue rebase later with 'git rebase --continue')  # d, drop <commit> = remove commit  # l, label <label> = label current HEAD with a name  # t, reset <label> = reset HEAD to a label  # m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]  # create a merge commit using the original merge commit's  # message (or the oneline, if no original merge commit was  # specified); use -c <commit> to reword the commit message  # u, update-ref <ref> = track a placeholder for the <ref> to be updated  # to this position in the new commits. The <ref> is  # updated at the end of the rebase  #  # These lines can be re-ordered; they are executed from top to bottom.  #  # If you remove a line here THAT COMMIT WILL BE LOST.  #  # However, if you remove everything, the rebase will be aborted.  # |

Changed the first lines as follows:

|  |
| --- |
| pick fb650d4 new ideas folder  squash 4b52fe5 taken up common.txt file  squash 26466a8 cleanup of documentation  squash 903ee39 more ideas in test1 |

+ save and exit

git will show the commit message from each commit involved in the rebase… (this comes in a file that can be edited (should be edited) and then saved and exited for the process to go on…)

|  |
| --- |
| # This is a combination of 4 commits.  # This is the 1st commit message:  new ideas folder  # This is the commit message #2:  taken up common.txt file  # This is the commit message #3:  cleanup of documentation  # This is the commit message #4:  more ideas in test1  # Please enter the commit message for your changes. Lines starting  # with '#' will be ignored, and an empty message aborts the commit.  #  # Date: Sat Apr 29 00:38:23 2023 +0200  #  # interactive rebase in progress; onto 3d3f585  # Last commands done (4 commands done):  # squash 26466a8 cleanup of documentation  # squash 903ee39 more ideas in test1  # No commands remaining.  # You are currently rebasing branch 'ideas' on '3d3f585'.  #  # Changes to be committed:  # new file: Dir\_test\_1/more\_ideas\_in\_dir\_test1.txt  # deleted: Documentation/GIT Cheat Sheet.docx  # deleted: Documentation/git-cheat-sheet-education.pdf  # new file: Documentation/placeholder  # new file: \_\_ideas\_branch/placeholder  # new file: common.txt  # |

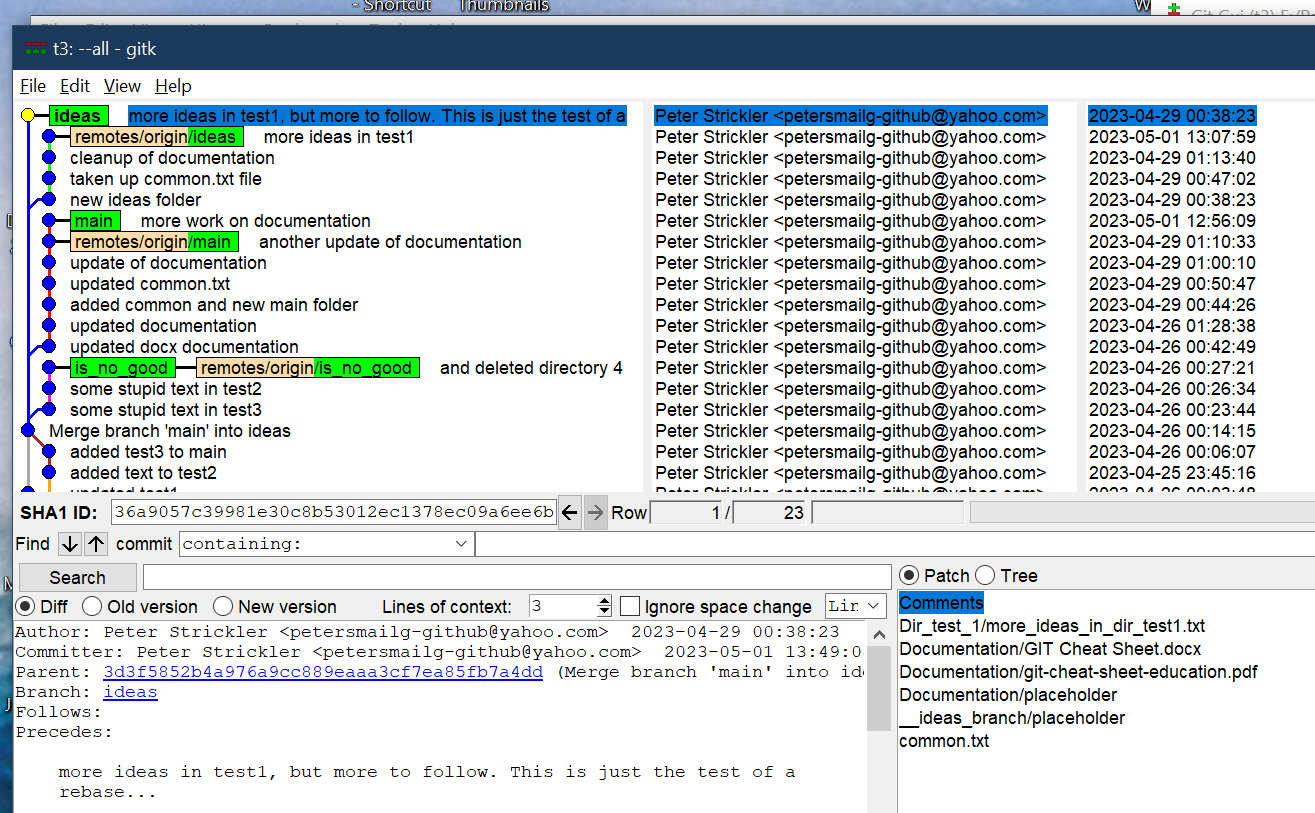
Changed the file as follows:

|  |
| --- |
| # This is a combination of 4 commits.  # This is the 1st commit message:  # This is the commit message #2:  # This is the commit message #3:  # This is the commit message #4:  more ideas in test1, but more to follow. This is just the test of a  rebase...  # Please enter the commit message for your changes. Lines starting  # with '#' will be ignored, and an empty message aborts the commit.  #  # Date: Sat Apr 29 00:38:23 2023 +0200  [and so on, the rest just stays the same as above…] |

:wq

|  |
| --- |
| PeterS@PCW10 MINGW64 /e/PeterS/t3 (ideas)  $ git rebase -i 3d3f585  [detached HEAD 36a9057] more ideas in test1, but more to follow. This is just the test of a rebase...  Date: Sat Apr 29 00:38:23 2023 +0200  6 files changed, 12 insertions(+)  create mode 100644 Dir\_test\_1/more\_ideas\_in\_dir\_test1.txt  delete mode 100644 Documentation/GIT Cheat Sheet.docx  delete mode 100644 Documentation/git-cheat-sheet-education.pdf  create mode 100644 Documentation/placeholder  create mode 100644 \_\_ideas\_branch/placeholder  create mode 100644 common.txt  Successfully rebased and updated refs/heads/ideas. |

|  |
| --- |
| PeterS@PCW10 MINGW64 /e/PeterS/t3 (ideas)  $ git log --oneline  36a9057 (HEAD -> ideas) more ideas in test1, but more to follow. This is just the test of a rebase...  3d3f585 Merge branch 'main' into ideas  42517a3 added test3 to main  4ab5490 updated test1  b84328a updating ideas  7ce1402 added text to test2  d0b4650 extended cheat sheet  f4c8867 new desktop.ini  fabe092 first commit |



|  |
| --- |
| PeterS@PCW10 MINGW64 /e/PeterS/t3 (ideas)  $ git status  On branch ideas  Your branch and 'origin/ideas' have diverged,  and have 1 and 4 different commits each, respectively.  (use "git pull" to merge the remote branch into yours)  nothing to commit, working tree clean |

Recap:

See the branch history:

git log –oneline

Get the original base of the ticket1 branch created from main:

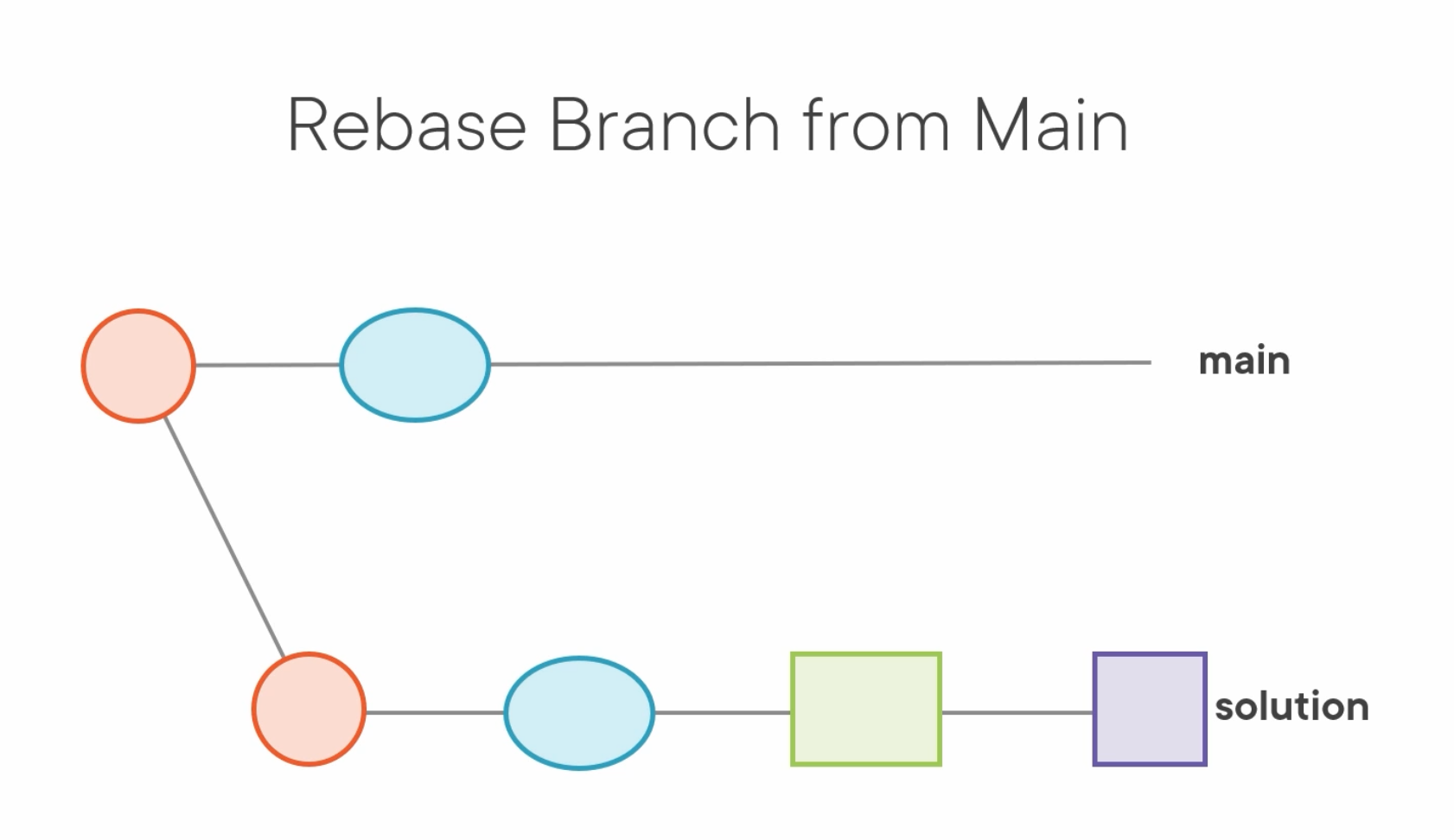
git merge-base ticket1 main

Start the (interactive) rebase from the commit sha

git rebase -i <commit-sha>

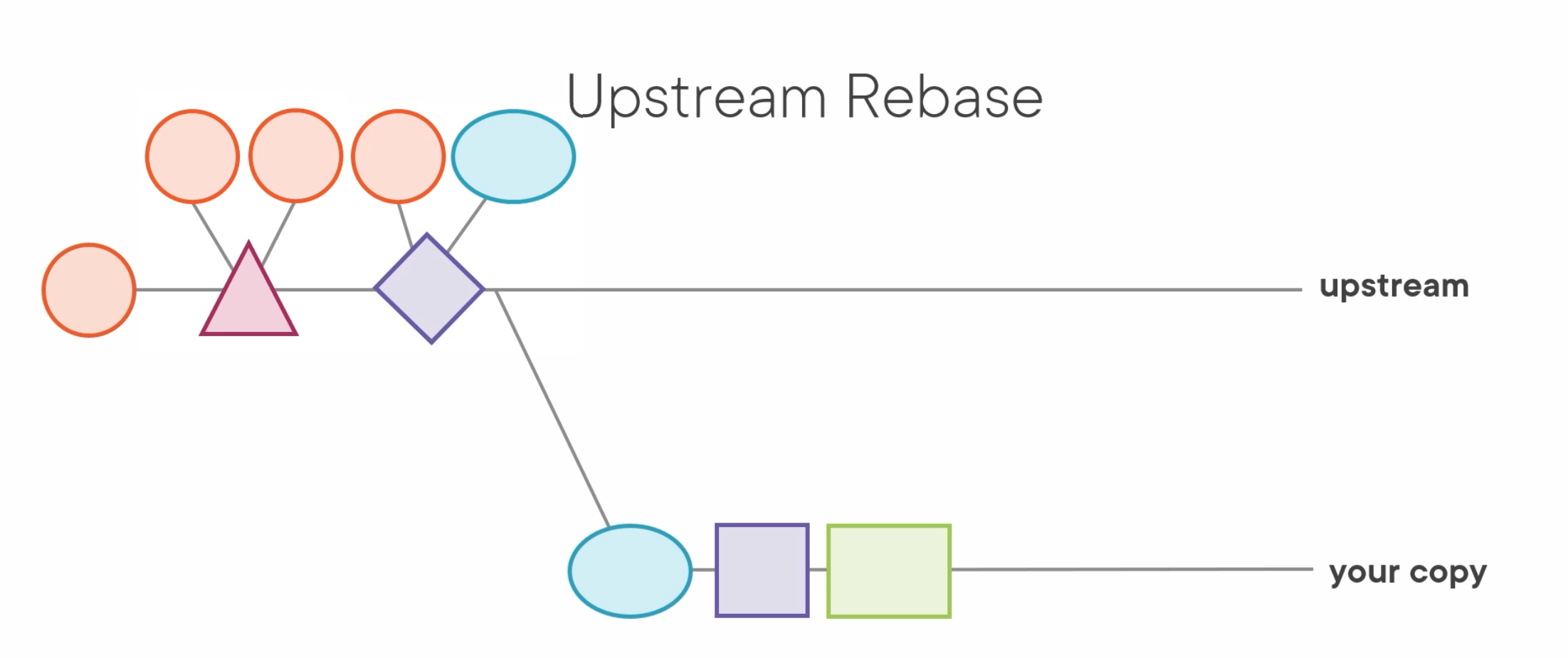
|  |
| --- |
|  |

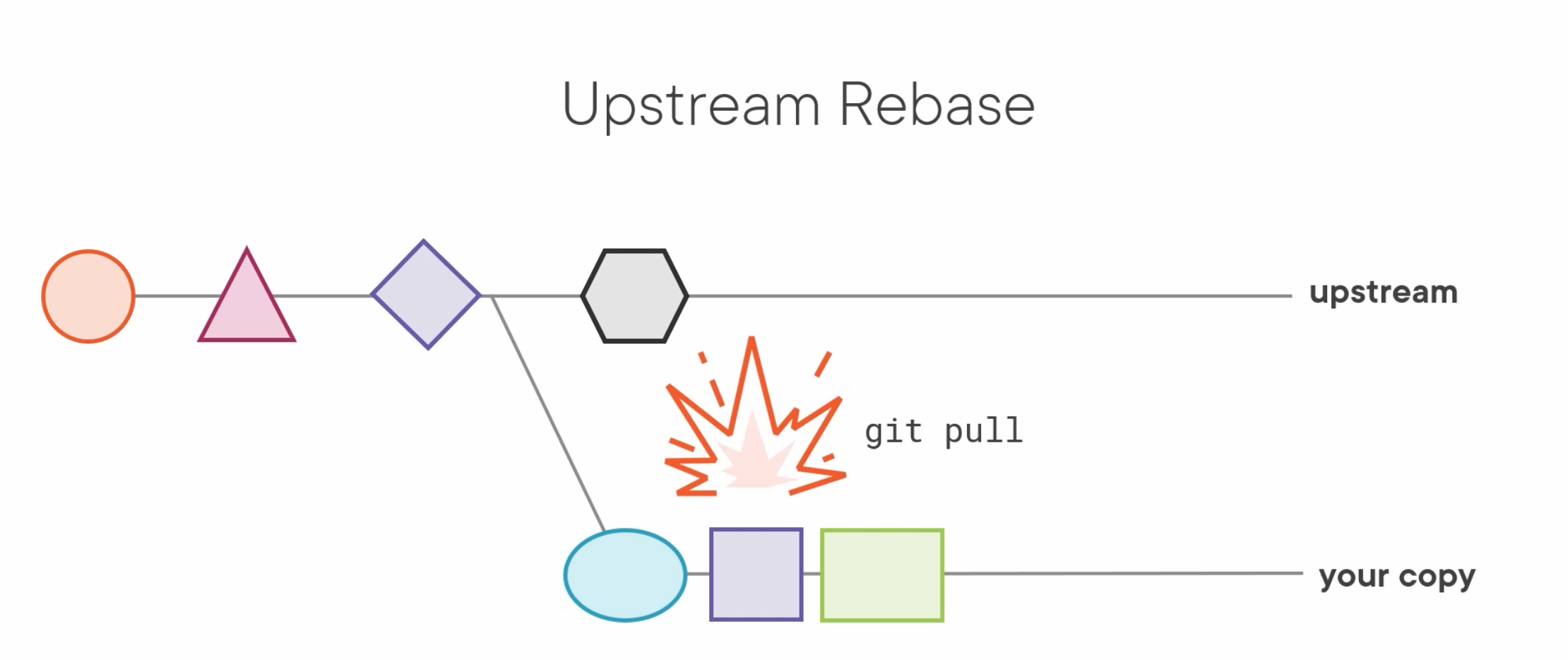
**Rebasing from main in (into) a feature branch**

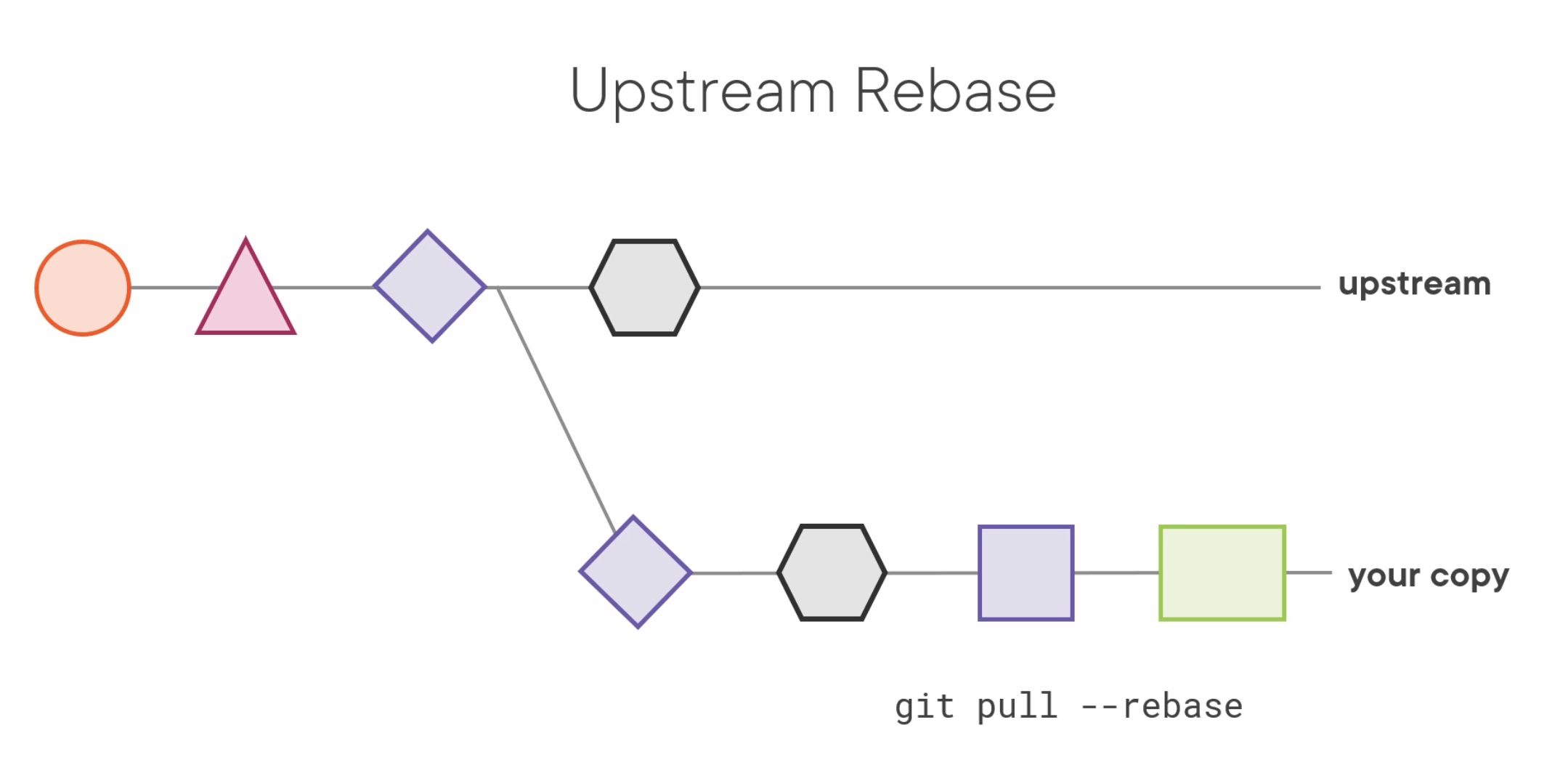


When in a branch, running git rebase main places all the changes made in main at the beginning of the branch and replays the later changes onto it (onto the last commit from main). Green and purple submits (from the image above) will be copied onto the blue circle commit, since they are changed (new parent). The original commits will eventually be garbage-collected.

Upstream Rebase







git pull --rebase

Uses reflog to look at each commit to find the common ancestor, even if it has been squashed, then sequence the commits in your branch.

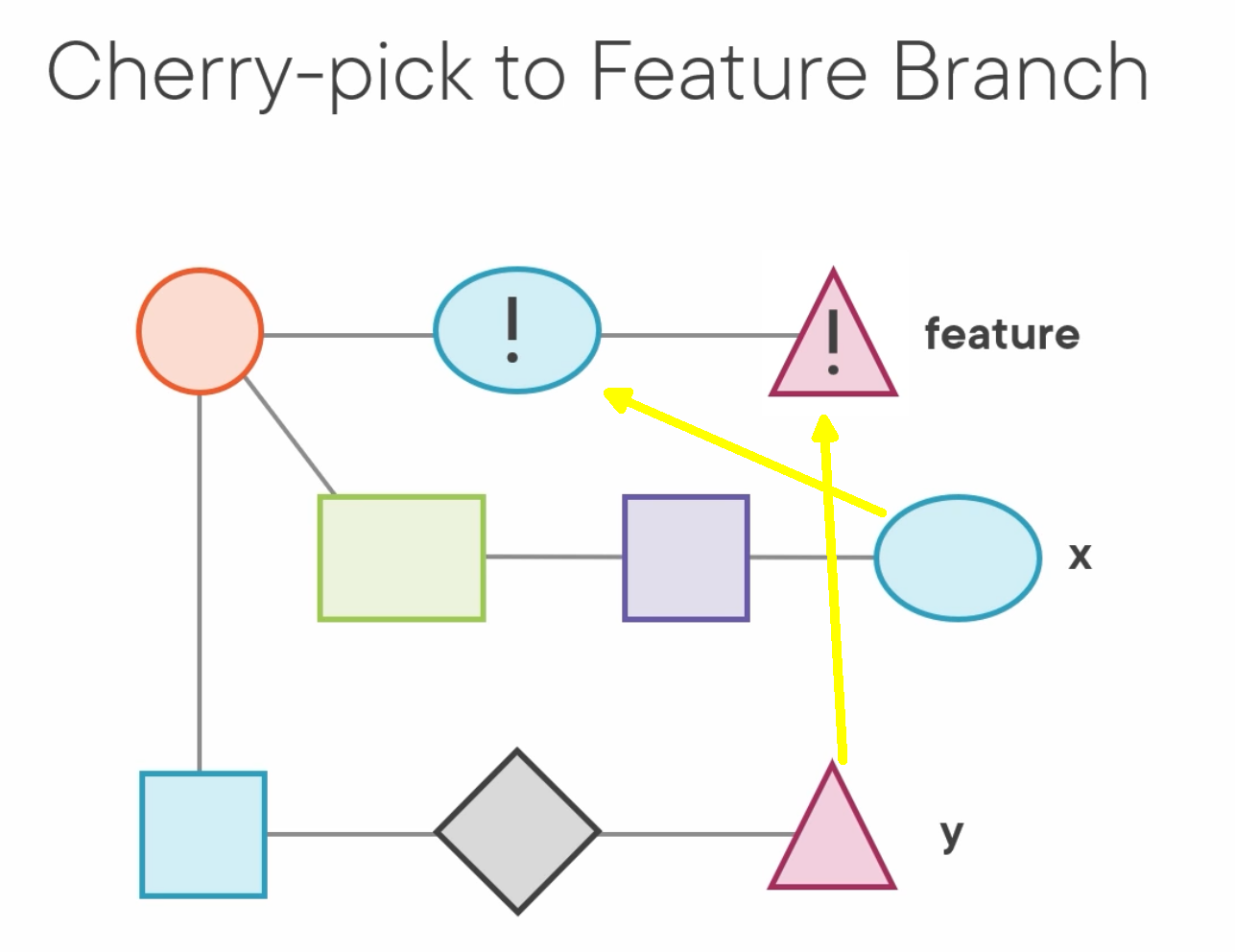
Hopefully youl’ll never need that …

**Cherry picking**

**Copy** specific commits to another branch. This means that the sha1 will be different in the branch where the cherry (commit) was picked (actually, copied from) and the one where the cherry was brought to. (after all, it is a **copy** of the original one)

git cherry-pick appends any commit to the current working HEAD of the branch you are in. Effectively making a copy of the commit in a different branch.

Common case in a bugfix that applies to multiple version of a product. (make the fix in one branch an cherry-pick it to the other branches where the fix also needs to be applied)



Sequence for cherry-picking

Find the commit you want

git log –oneline

List the log from another branch

git log <branch-name> --online

Where do you want to put the commit ? Checkout that branch…

git checkout <branch-name>

Perform the cherry-pick to append the specified commit to HEAD of the current branch.

git cherry-pick <commit-sha>

**How to force a push request in Git:**

This command will force a push request. This is usually fine for pull request branches because nobody else should have cloned them.  
But this isn't something that you want to do with public repos.

git push -f

**Conclusion**

These commands can dramatically improve your productivity in Git. You don't have to remember them all – that's why I have written this cheat sheet. Bookmark this page for future reference or print it if you like.

**Renaming a branch**

**To do this, use the following steps:**

1. Switch to the master via the command “git checkout master”.
2. Now enter the following command if you want to rename a Git branch: “git branch -m old-name new-name”.
3. To ensure that the rename was successful, retrieve the current status of the branch using the “git branch -a” command.

**Losing your HEAD**

Normally, HEAD is a reference to a branch which in turn is a reference to a commit.

But you can also directly checkout a commit, without creating a new branch. This lets HEAD point at the commit and anything that is done then will not affect the branch, but will be handled as floating HEAD.

Git log -> for SHA1 of commit

commit 3d3f5852b4a976a9cc889eaaa3cf7ea85fb7a4dd (HEAD -> main, ideas)

git checkout 3d3f58

Note: switching to '3d3f58'.

You are in 'detached HEAD' state. You can look around, make experimental

changes and commit them, and you can discard any commits you make in this

state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may

do so (now or later) by using -c with the switch command. Example:

git switch -c <new-branch-name>

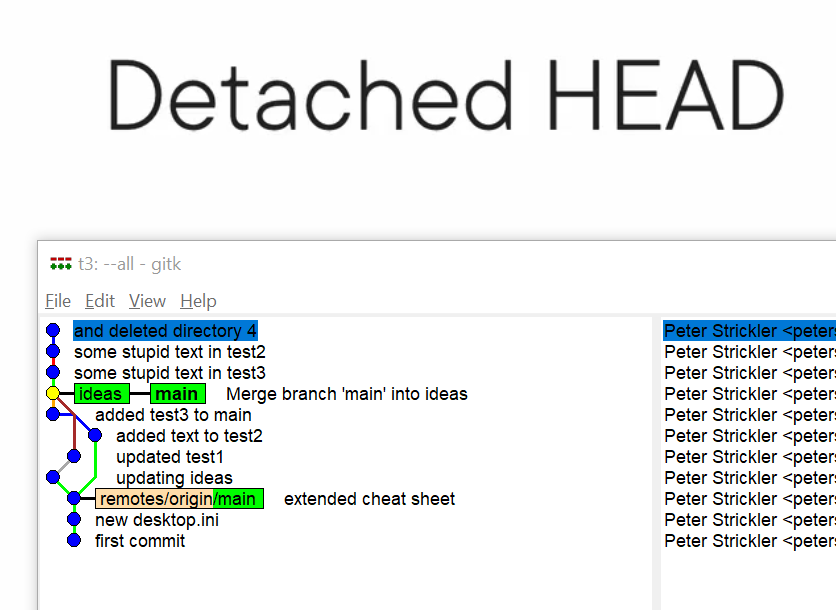
Or undo this operation with:

git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 3d3f585 Merge branch 'main' into ideas

PeterS@PCW10 MINGW64 /e/PeterS/t3 ((3d3f585...))



Pulling the HEAD back to main:

git switch main (but this leaves the commits without having a branch there as well somehow dangling in the nothing and more or less ready to be garbage collected…)

$ git switch main

Warning: you are leaving 3 commits behind, not connected to

any of your branches:

2a61f9d and deleted directory 4

820c6f0 some stupid text in test2

ccf7f50 some stupid text in test3

If you want to keep them by creating a new branch, this may be a good time

to do so with:

git branch <new-branch-name> 2a61f9d

Switched to branch 'main'

Your branch is ahead of 'origin/main' by 5 commits.

(use "git push" to publish your local commits)

If those detached HEAD commits should be kept anyway, we have to move the HEAD back to the one we want to keep via its sha1 (as long as we still have it)

$ git checkout 2a61f9d

Note: switching to '2a61f9d'.

You are in 'detached HEAD' state. You can look around, make experimental

changes and commit them, and you can discard any commits you make in this

state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may

do so (now or later) by using -c with the switch command. Example:

git switch -c <new-branch-name>

Or undo this operation with:

git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 2a61f9d and deleted directory 4

And then define a new branch where the HEAD is

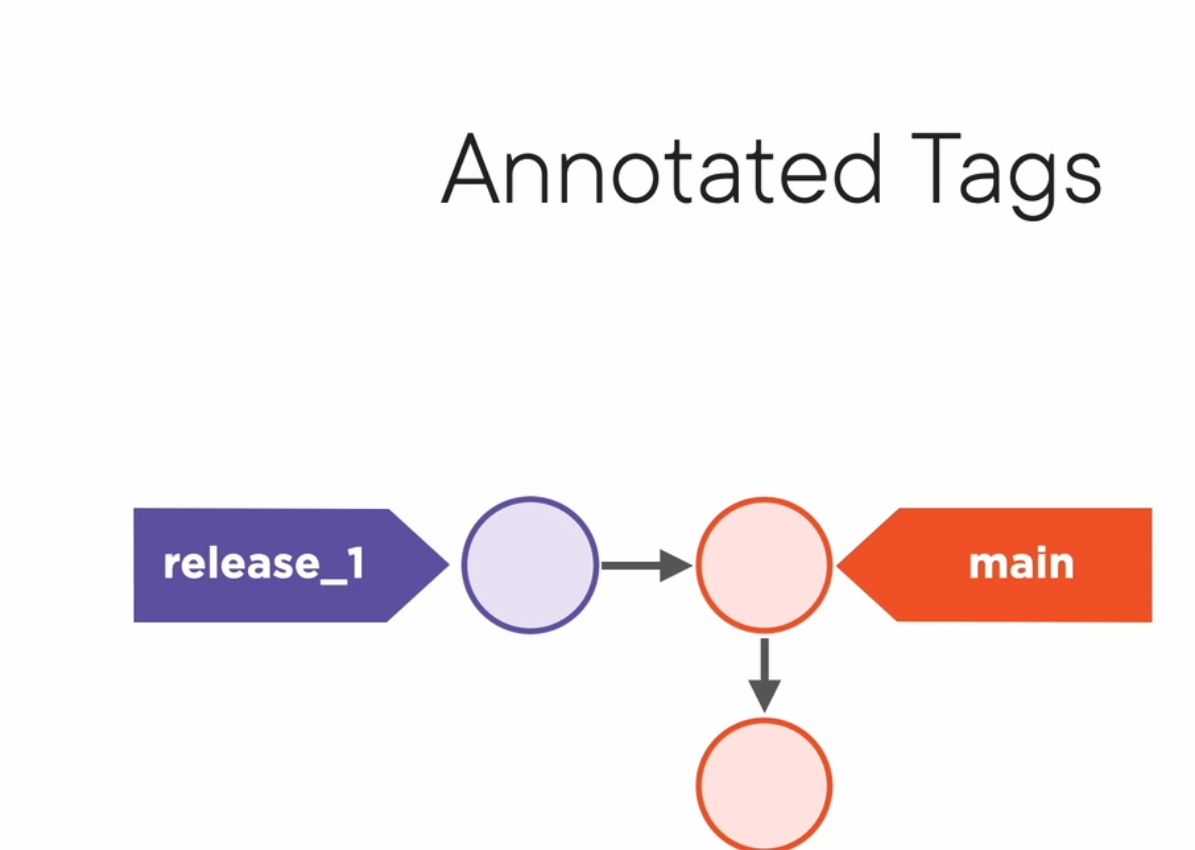
Git branch nogood

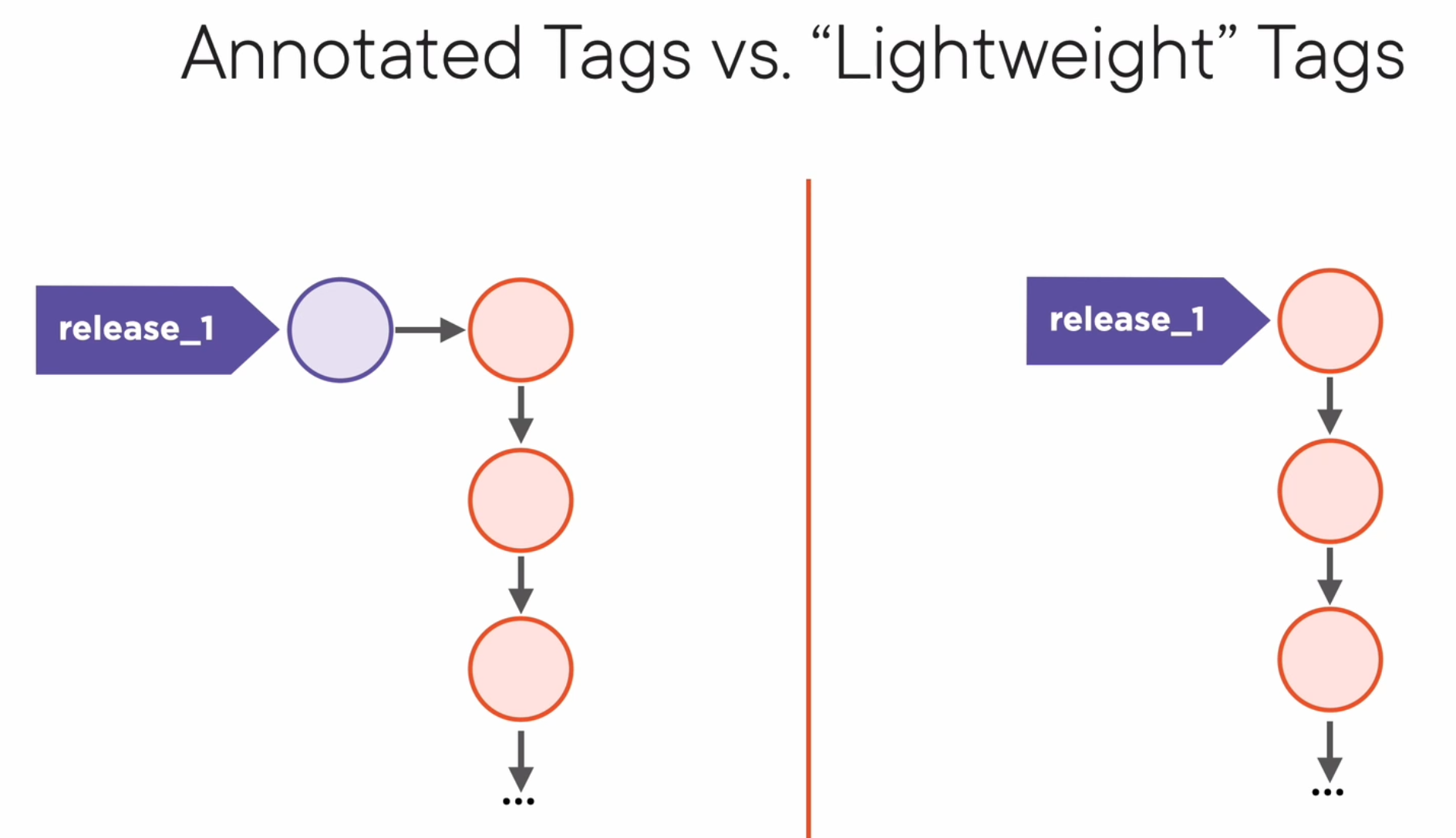
**Tags and annotated tags**

Git tag release\_1 🡨 simple tag

Git tag release\_1 -a -m “first release, still unstable” 🡨 annotated tag ( -a ) with a message to it. (this is one of the basic components in the git data base (the others being blobs, trees and commits)

A tag is a database object itself, that in turn points to a commit.





**==> Setting up a new branch**

git checkout -b new\_name

git switch -c new\_name

**==> Renaming a branch**

git branch -m old\_name new\_name

**==> Renaming the active branch**

git branch -m new\_name\_for\_the\_current\_branch

**==> Deleting a branch**

git branch -d branch\_name

- branches with uncommitted work cannot be deleted, unless you force git to do so ( -D )

git branch -D branch\_name

- the current branch cannot be deleted (switch to another one first...)

**==> Merging branches**

To merge branch a into branch b, fist switch to the target branch

(

git switch b (target branch)

git merge a (source branch)

)

i.e. merge feature (back) into main:   
git switch main

git merge feature

In case a merge comes up with conflicts, and needs to be aborted:

Stop the merge and return directory to the previous state before the merge started:

git merge --aborted

'Evil merge' : a merge that introduces new content in a merge

a merge commit should only combine existing content (i.e. do not change other stuff (that was not conflicting or introduce new stuff) when cleaning up conflicts.)

-> abort merge to commit new content (in a new commit), then restart the merge

remember

git log --oneline

**==> Comparing branches**

git diff

shows what has changed that is not staged for a commit

git diff --cached

shows what is staged for commit that is different from the last commit

in other words, what would be committed if you ran <git commit>

gif diff HEAD

will show what has changed since the last commit

or what would be committed, if <git commit -a> were run

git diff -w

to ignore white space differences.

git diff <commit sha1>

shows the difference between the specified commit and the current working directory

git diff --cached <commit sha1>

shows the difference between the specified commit and what is staged

git diff <commit1 sha1> <commit2 sha1>

shows the difference between the two specified commits

git diff branch\_a branch\_b

shows the difference between the tips of the two specified branches

git diff branch\_a...branch\_b

shows what has changed in branch\_b since branch\_a was started off of it.

git diff feature...main

shows what has changed in main since feature was started off of it.

git diff feature main file.txt

shows the difference of a file in the two specified branches

**==> Looking at specific files**

git show <sha1 of the file>

sha1 e.g. from the output of a git diff

**===> Fork (in GitHub)**

go to an existing repo and click on 'fork' (upper right), specify where to fork to...

then clone the repo onto your local machine

git clone https://github.com/seethatgo/widgit.git

git remote -v

shows the ULR for the repo where the project was forked (or cloned) into.

list remotes and URLs

'origin' is the default name for the remote server

git remote add <name> <remote-url>

provide the name for a remote server (???)

git fetch

gets all information from remote that isn't already in the local copy - you can then evaluate it and decide what you want to do...

git pull

is a combination of fetch and merge...

git push

will only work if there are no conflicts between the local version and the remote.

push the feature4 branch to remore and set origin as the upstream branch

git push -u origin feature4

list branches on remote

git ls-remote

git remote update

updates the information about the status on the remote.

check with <git status> afterwards

Get the feature4 branch from remote to local:

git fetch origin feature4

Check the branch that have been downloaded

list both remote-tracking and local branches

git branch -a

Set up a local branch to track the remote branch

git checkout --track origin/feature4

**===> Pull request**

when you don't have the rights to merge directly into main, the pull request tells the maintainers of the project to review and then merge the new code.

a) push branch to remote

b) when ready, open a pull request

c) add reviewers & discuss changes

d) Make fixes in local branch

e) ush to remote and it will automatically (?) be added to the pull request.

--> do not rebase after pushing to remote!!

**===> .gitignore**

<https://github.com/github/gitignore>

\* ? ! / [a-zA-Z]

anything, one character, negator, directory separator, range

# this is a comment

# for comments

\*\*/bin

\*\* matches any directory in the repository

\*.zip

\* matches any file in the repository

/bin

relative to .gitignore directory (put a .gitignore at your project root!)

git rm --cached <filename>

delete file from git repo

git rm <filename>

delete file from git repo and local filesystem.

.git/info/exclude

ignore patterns for your system only. (the exclude file is set up just like .gitignore, but it's only for the own system.)