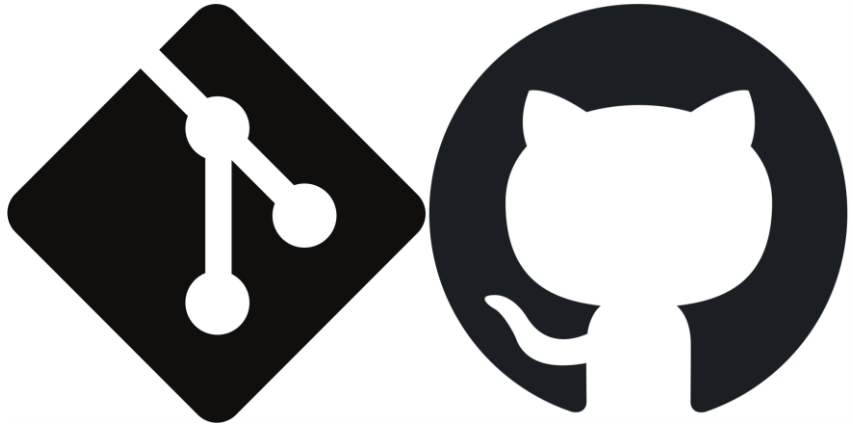


Introduction to Git & GitHub

Jennifer Graham
Tiago Silva
David Ryder
Stephen Gregory

Session 2
6th June 2024

Managing your own projects

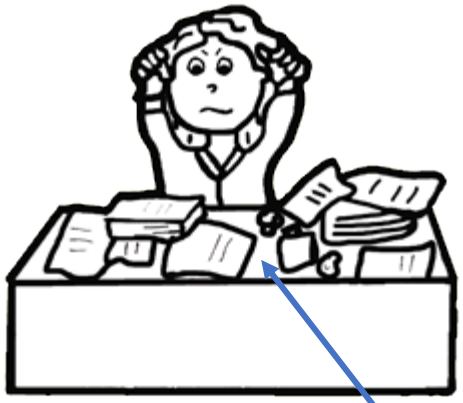
When mistakes happen

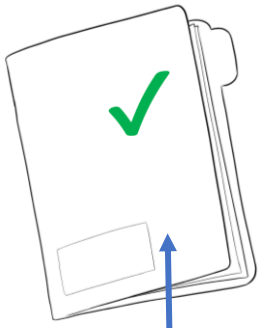
`git checkout`

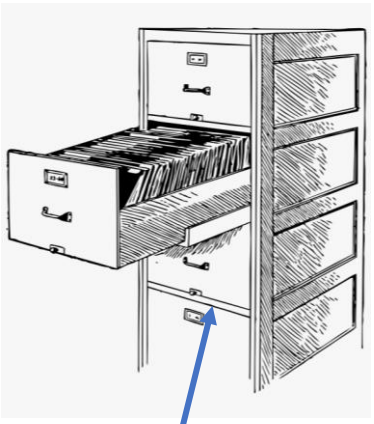
`git reset/restore`

`git revert`

But how does git *really* work?







Working directory	Staging area	Commit tree
Messy!	Changes to include at next save	All the saved versions

git as your assistant:

You make requests and git modifies the files and directories in your local directory.

"Get me yesterday's version of this chapter"

checkout yesterday's file

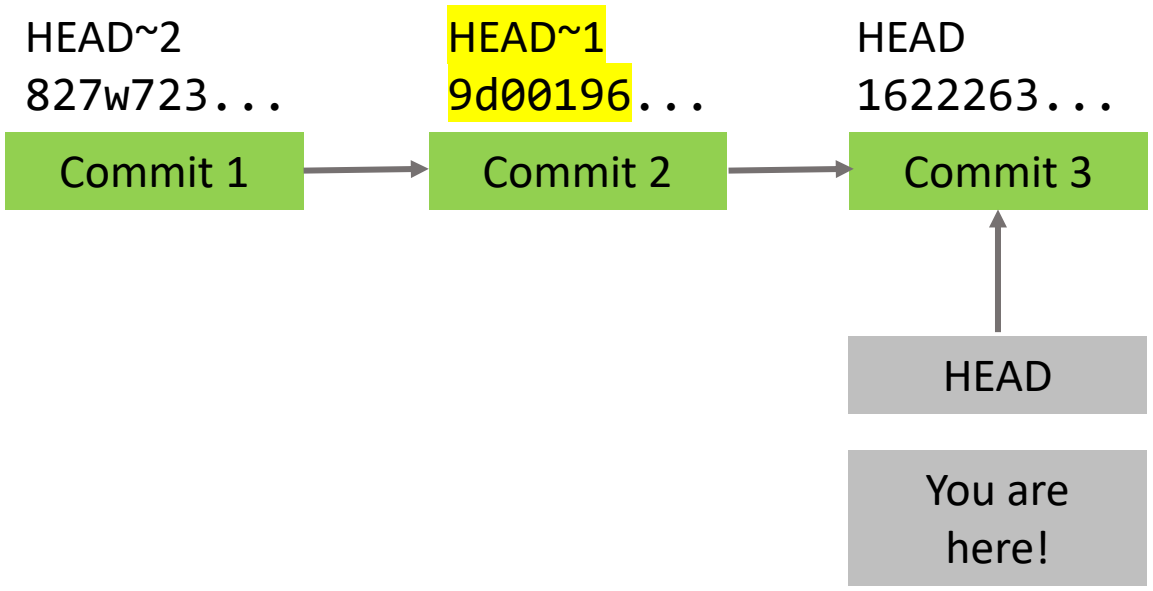
"I will want to keep this document at the next save"

add file to staging area

"Save all the files marked"

commit (staged files)

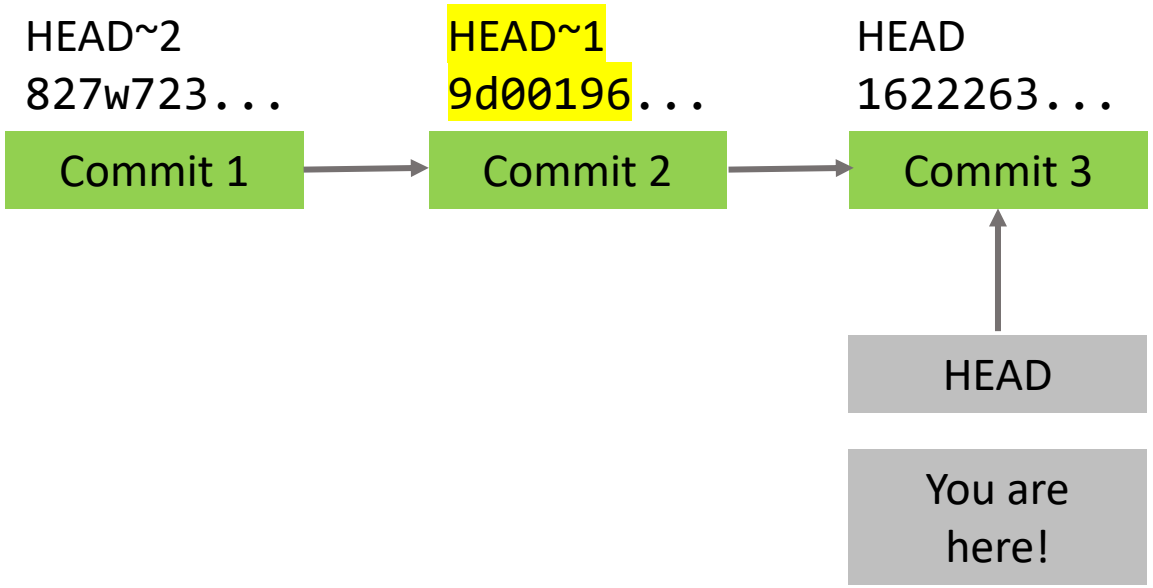
How to refer to a `commit` (version)



How to refer to a `commit` (version)

Relative to the last commit

- HEAD is last commit
- HEAD~*n* is *n* before last



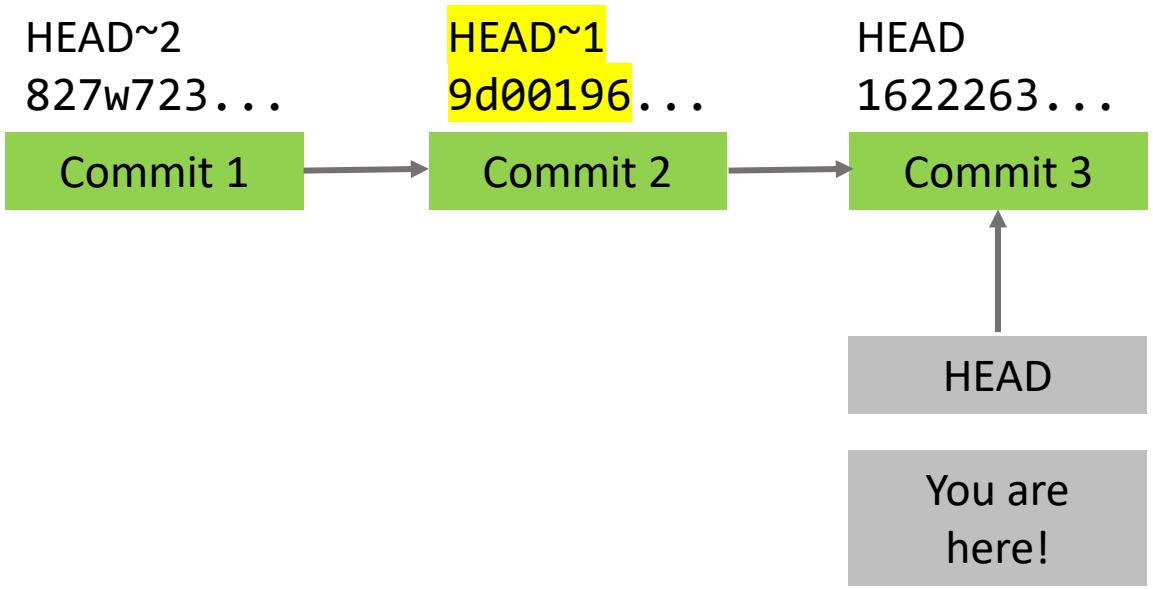
How to refer to a `commit` (version)

Relative to the last commit

- HEAD is last commit
- HEAD~*n* is *n* before last

Absolute reference

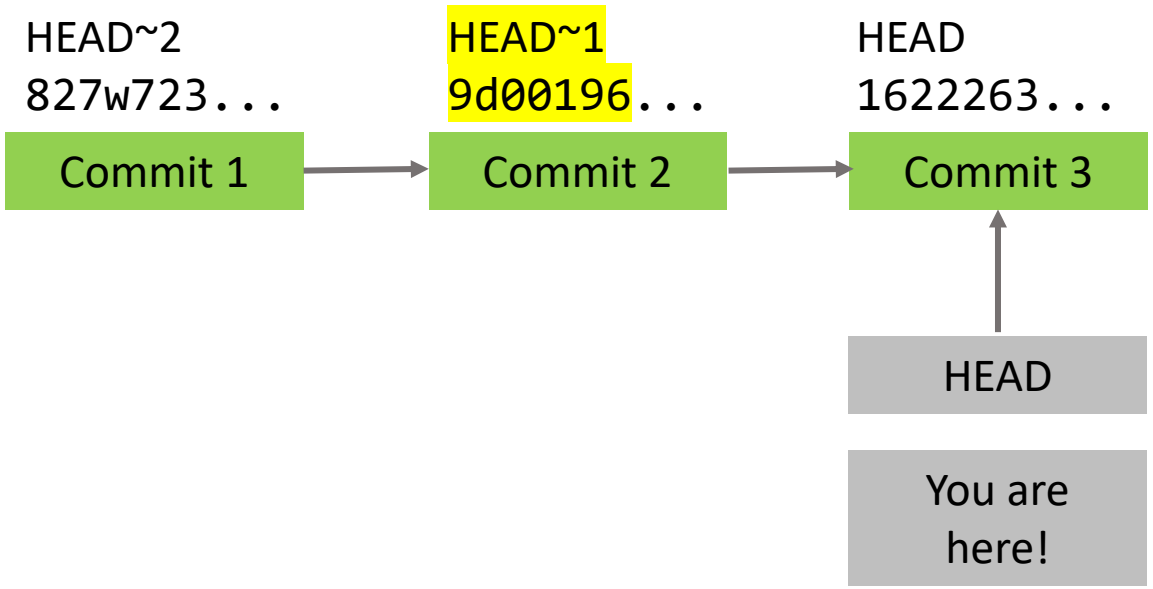
- use a hash from `git log`



How to refer to a `commit` (version)

Relative to the last commit

- HEAD is last commit
- HEAD~*n* is *n* before last



Absolute reference

- Use a hash from `git log`
- The first 10 characters are usually enough

```
$ git log
commit 1622263b54404019a9548d7a9a2e2e81992c1cd7
Author: tiagoams <tiago.silva@cefas.co.uk>
Date:   Mon Jul 1 12:53:44 2019 +0100

    Commit 3

commit 9d00196f3335fcbab94d0f6cd4af6f6ebaf663f7
Author: tiagoams <tiago.silva@cefas.co.uk>
Date:   Mon Jul 1 12:51:13 2019 +0100

    Commit 2
```

Restore previous version of file

`git checkout [<commit>] [--] <path>`

"--" just means staying in the same branch. Can be used to avoid confusion with checking out a branch of the same name as a file.

Working directory	Staging area	Commit tree
Yes	**Yes**	No

Restore previous version of file

`git checkout [<commit>] [--] <path>`

"--" just means staying in the same branch. Can be used to avoid confusion with checking out a branch of the same name as a file.

Working directory	Staging area	Commit tree
Yes	**Yes**	No

```
# Added 2 commits, ver. n and n+1
$ git checkout HEAD~1 README.md
$ cat README.md
#     Commit 2
$ git checkout HEAD README.md
$ cat README.md
#     Commit 3
$
```

Restore previous version of file

`git checkout [<commit>] <path>`

Working directory	Staging area	Commit tree
Yes	**Yes**	No

```
# Added 2 commits, ver. n and n+1
$ git checkout HEAD~1 README.md
$ cat README.md
#     Commit 2
$ git checkout HEAD README.md
$ cat README.md
#     Commit 3
$
```

- What would happen here?

```
git checkout HEAD~1 README.md
git commit
```

Restore a previous commit (all files)

`git checkout [<commit>] [--force]`

All files will be set to earlier versions.

*You will be stopped from doing this if you have uncommitted changes in your files. --force will override existing changes.

Working directory	Staging area	Commit tree
Yes*	**Yes***	No

```
$ git checkout HEAD~1

error: Your local changes to the following files
   would be overwritten by checkout:

        README.md

Please commit your changes or stash them before you
   switch branches.

Aborting

$
```

- We will return to this when we talk about Detached HEAD state.

Try this! (10 mins)

- Open the editor and create a new file `poem.txt`

(right click on directory and choose New Text Document)

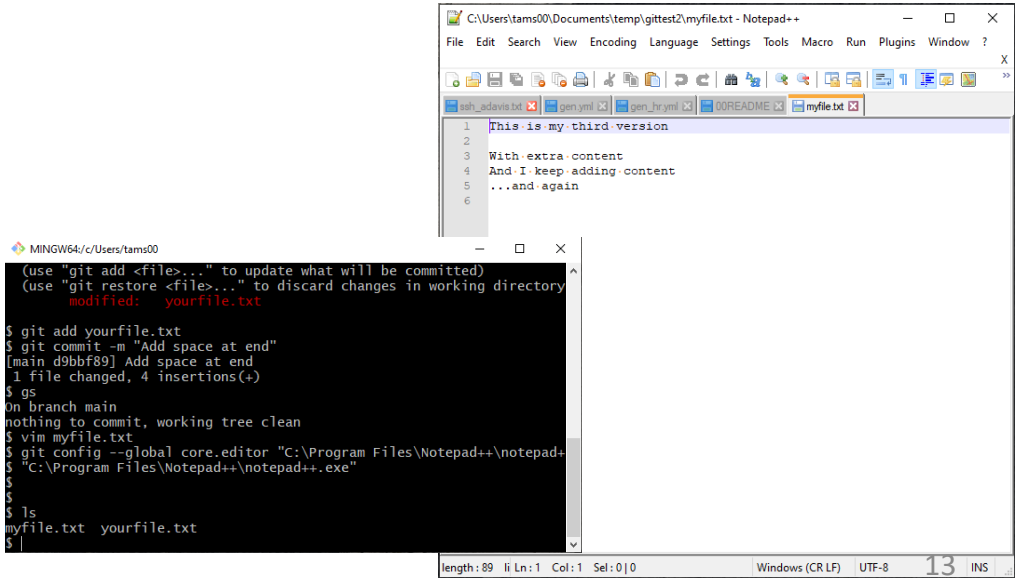
- Copy the first verse of a poem and save.
- git commit -m "add first line".
- Add another verse and commit.
- And repeat again.

Ex. 1

1. Retrieve previous version of poem.

   NOTE: Keep the editor open and see what happens to the file (Notepad requires reopening the file).

2. Retrieve the version before that.
3. Retrieve the most recent version.



Reset a git add (undo staging or tracking file)

`git reset [<path>]`

new: `git restore --staged <path>`

```
•   $ git add myfile.txt
•   $ git status
•   On branch main
•   Changes to be committed:
•     (use "git restore --staged <file>..." to unstage)
•           modified:   myfile.txt

•   $ git reset myfile.txt
•   Unstaged changes after reset:
•   M       myfile.txt
•   $ git status
•   On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   myfile.txt

no changes added to commit (use "git add" and/or "git commit -a")
$
```

Working directory	Staging area	Commit tree
Can do with --hard	**Yes**	No

Undo file changes since last commit

```
git reset --hard                          # All files
```

new: `git restore --source=HEAD [<path>]  # One or more files`

- Go back to previous commit, discarding all changes in working directory

Working directory	Staging area	Commit tree
Yes	Can do	No

```
$ rm myfile.txt                           # Mistake
$ ls                                      # Oh noooooo!
yourfile.txt

$ git reset --hard                        # Discard changes since last commit

HEAD is now at d9bbf89 Add space at end

$ ls

myfile.txt  yourfile.txt                  # Uff! The file was recovered
$
```

The split personality of reset

`git reset --hard`

- Rolls back to HEAD discarding staged and unstaged changes

```
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   new_file.txt

$ git reset --hard
HEAD is now at b22bc60 added even 1 more echo
$
```

Working directory	Staging area	Commit tree
Yes	**Yes**	**Yes**

`git reset <path>`

- Unstage files
- Doesn't modify working directory or commit history

Working directory	Staging area	Commit tree
No	**Yes**	No

The split personality of reset

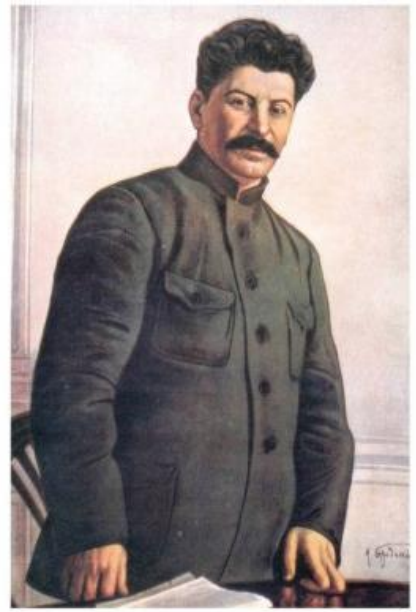
`git reset --hard <commit>`

- Rolls back to <commit> discarding later commits

```
$ git commit test_echo.sh -m "This might be a bad
commit..."

[main 136eb6e] This might be a bad commit...

1 file changed, 1 insertion(+)

$ git reset --hard HEAD~1

HEAD is now at b22bc60 added even 1 more echo
$
```

Working directory	Staging area	Commit tree
Yes	Yes	Yes

Rewriting history?



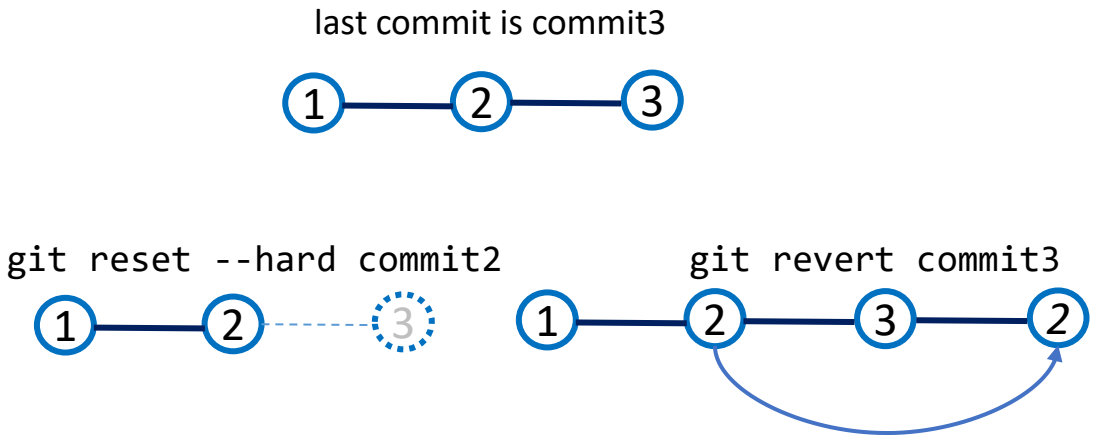
Undo changes of single commit

`git revert HEAD`

- Generate a new commit reversing the changes made in last commit
- Maintains offending commit in the history

Working directory	Stagging area	Commit tree
Yes	No	**Yes (preserving history)**

```
$ touch newfile
$ git add newfile
$ git commit -m "Added empty file"
[rev a6ef2bd] Added empty file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 newfile
$ git revert HEAD
[rev 1c00b28] Revert "Added empty file"
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 newfile
$ git log
commit 1c00b28a304f93e9a4f3fddc8b5df296ed69f474 (HEAD -> rev)
Author: tiagoams <tiago.silva@cefas.co.uk>
Date:   Wed Sep 4 15:25:58 2019 +0100

    Revert "Added empty file"

    File not needed. Better do this using oldfile.
```



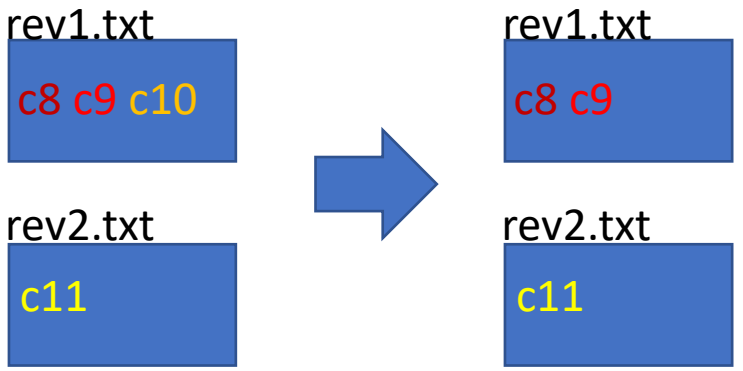
Undo changes of single commit

`git revert <commit>`

- Can only revert independent commits
- Let's revert c10
- Reverting c9 would not have been trivial as c10 adds to c9

Working directory	Stagging area	Commit tree
Yes	No	**Yes (preserving history)**

```
$ cat rev1.txt
c8 c9 c10
$ cat rev2.txt
c11
$ git revert HEAD~1
[rev ad6a74e] Revert "c10"
 1 file changed, 1 insertion(+), 1 deletion(-)
$ cat rev1.txt
c8 c9
$ cat rev2.txt
c11
$
```



Summary

Clearing up mistakes
Restore previous version of file
`git checkout [<commit>] <path>`
Go back to last commit, discarding all changes in working directory Discard all commits after `[<commit>]`
`git reset --hard HEAD` `git reset --hard [<commit>]`
Undo git add (undo staging or tracking file)
`git reset <path>`
Undo changes introduced by single commit, recording this in history
`git revert <commit>`

Undo this! (10 mins)

- Create a file with the beginning of a poem and add and commit it
- Add the rest of the poem and make a 2nd commit
- Modify the file adding your own changes to the poem. Stage the file.

Ex. 1 - Unstage the file. Check if the contents have changed.

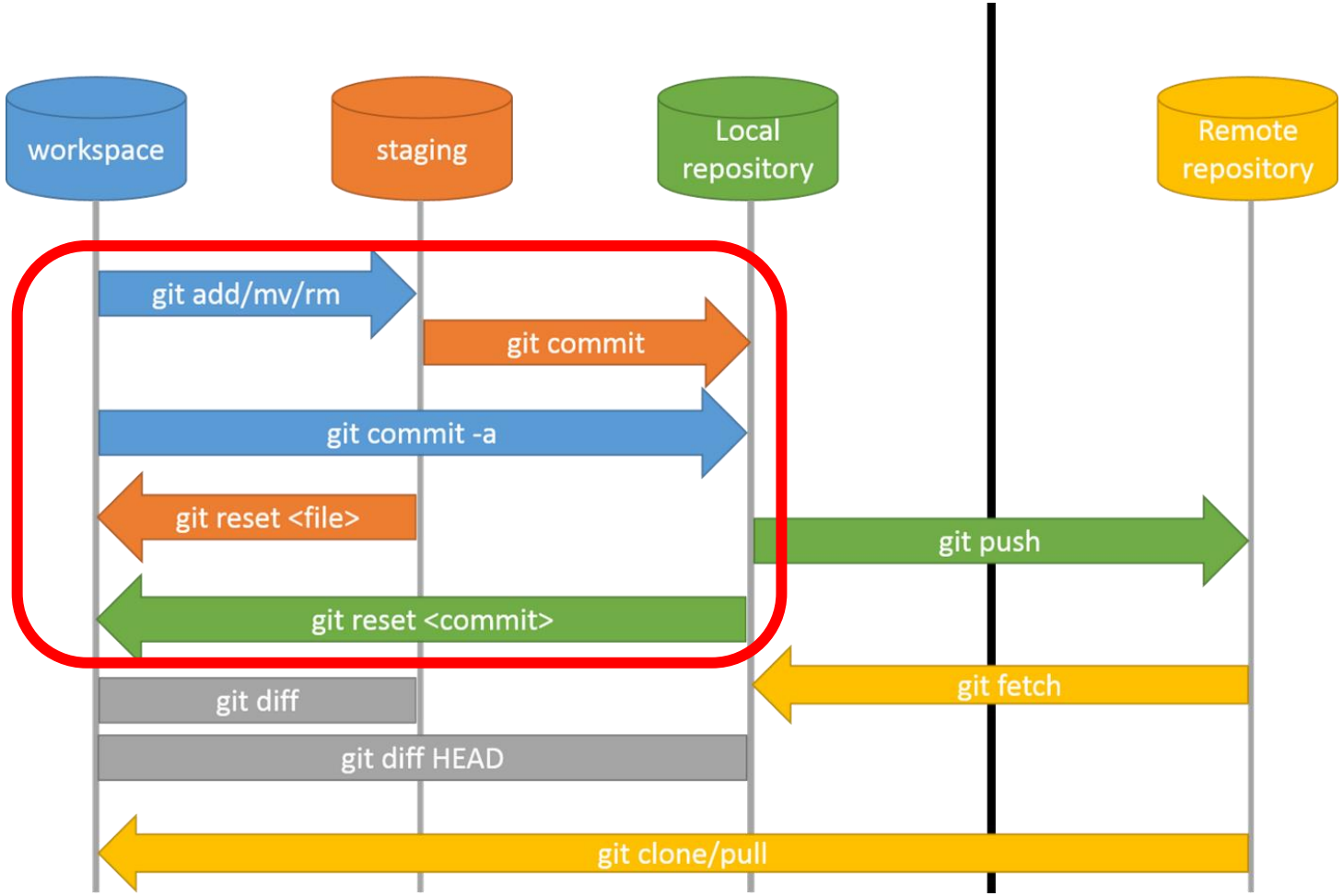
Ex. 2 – Clear any changes since last commit.

- Now modify the poem and create a 3rd commit

Ex. 3 - Undo the last commit keeping the history. Inspect the history.

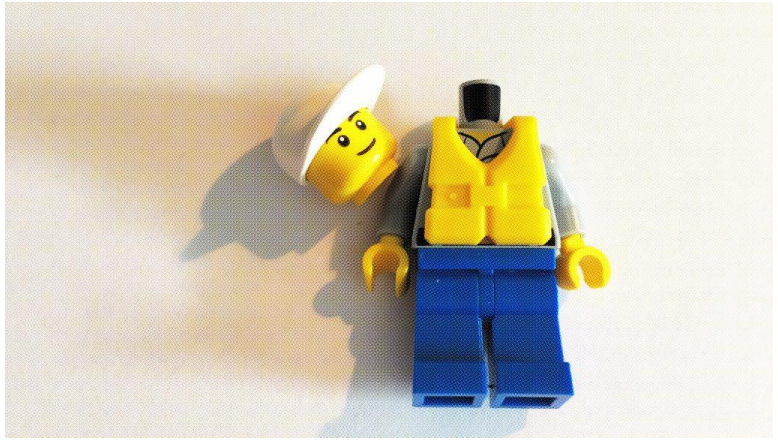
```
Because I could not stop for Death
by Emily Dickinson

Because I could not stop for Death,
He kindly stopped for me;
The carriage held but just ourselves
And Immortality.
```

Local vs. remote

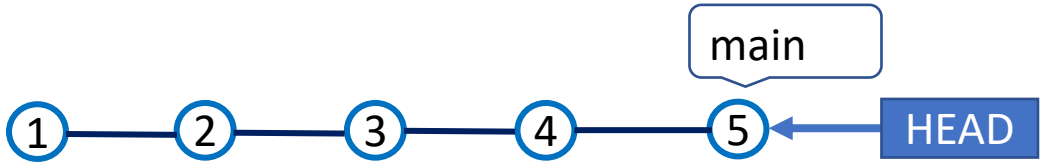


Where is my mind?

Detached HEAD state

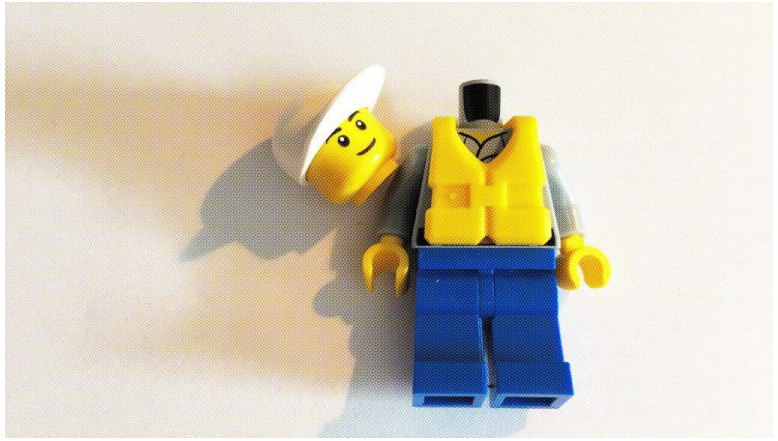


- HEAD is a pointer that points to the end of the branch*
- When you checkout an intermediate commit, the HEAD gets “detached”

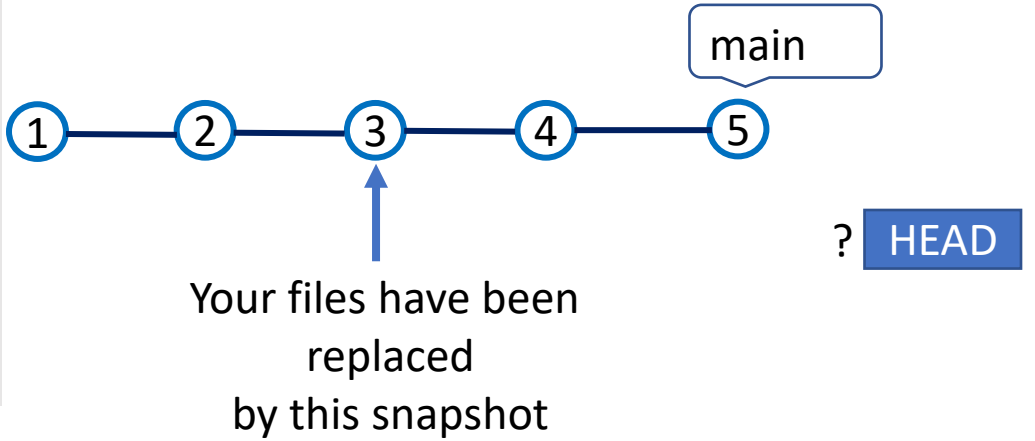


Where is my mind?

Detached HEAD state



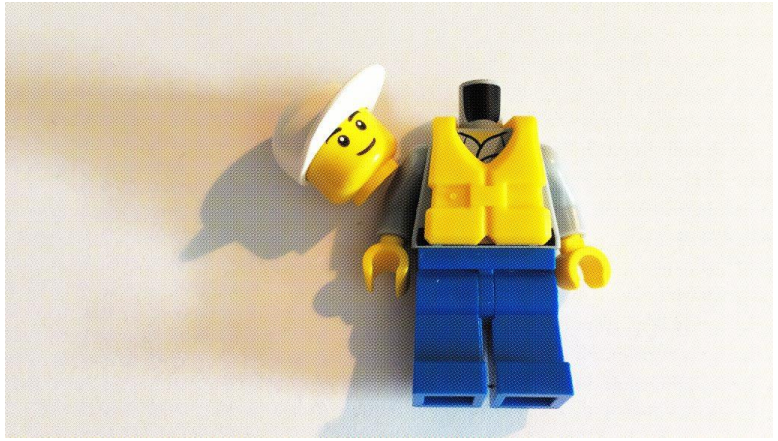
```
$ git checkout 37fdafddc299ab6d4e8ccf3400c6b2b82acdfb6c
Note: checking out '37fdafddc299ab6d4e8ccf3400c6b2b82acdfb6c'.

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 performing another checkout.
(...)
HEAD is now at 37fdafd Revert "Rubbish commit“
$
```



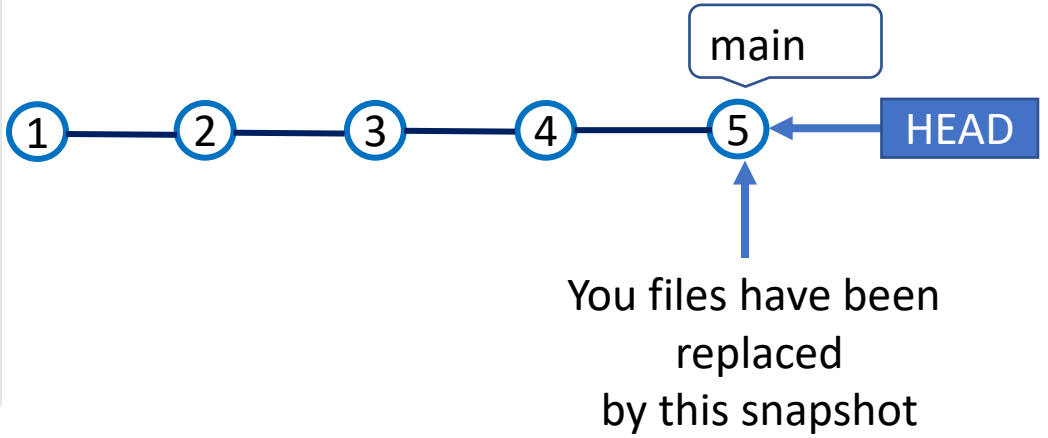
This is a normal state if you are inspecting past commits, but don’t make changes as these can be easily lost.

Where is my mind?

Detached HEAD state



```
$ git checkout 37fdafddc299ab6d4e8ccf3400c6b2b82acdfb6c
Note: checking out '37fdafddc299ab6d4e8ccf3400c6b2b82acdfb6c'.

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 performing another checkout.
(...)
HEAD is now at 37fdafd Revert "Rubbish commit“
$ git checkout main
Previous HEAD position was 37fdafd Revert "Rubbish commit"
Switched to branch ‘main’
$
```



Restore order by checking out (the end point of) your branch again.

Time for a break!

https://www.selectcourtreporters.com/how-to-stay-active-while-working-from-home/

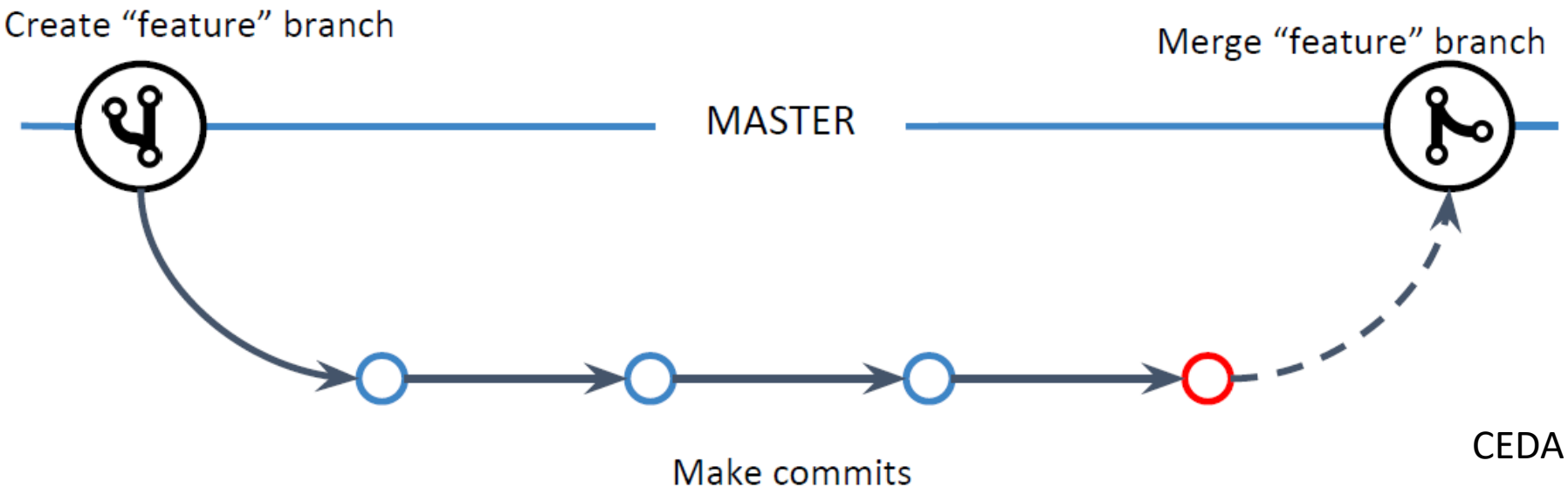
Branching and Merging

- What is a branch?
- When should I create a branch?
- How do I merge changes?

```
git branch
git checkout
git merge
```

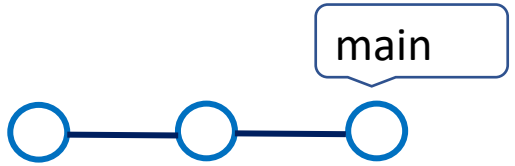
Branching

- The main branch should always work.
- To test changes or new features, create a branch.
- A new branch creates a new end point in your chain of changes



Branching

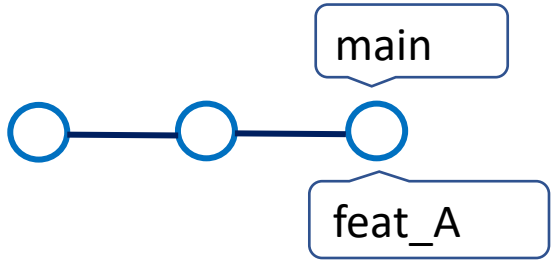
- Branches are pointers to an end commit



Note: main is just the default name for the first branch

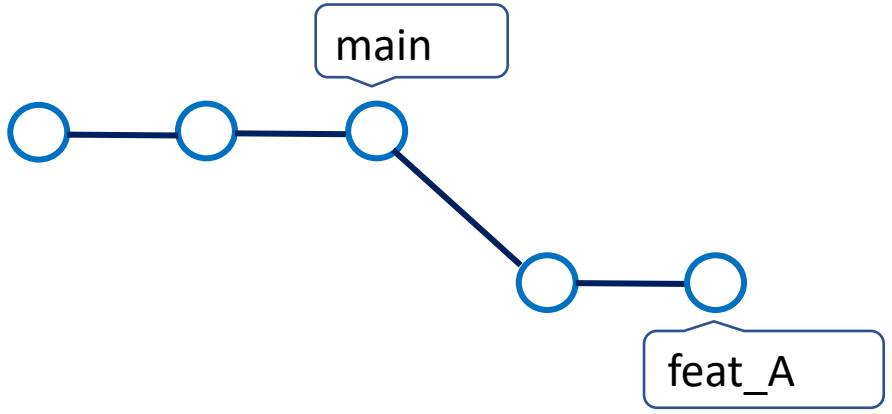
Branching

- Branches are pointers to an end commit
- By creating branch feat_A new end point is created



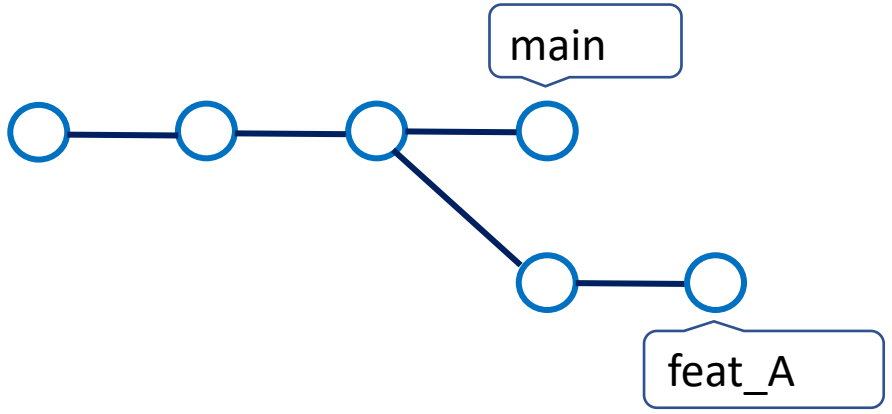
Branching

- Branches are pointers to an end commit
- By creating branch feat_A new end point is created
- New commits in feat_A won't affect main



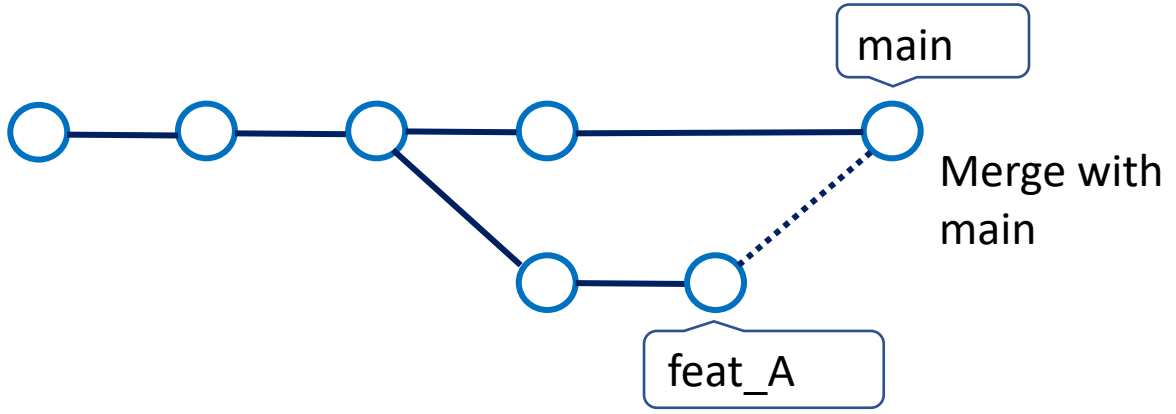
Branching

- Branches are pointers to an end commit
- By creating branch feat_A new end point is created
- New commits in feat_A won't affect main
- And new commits in main won't affect feat_A



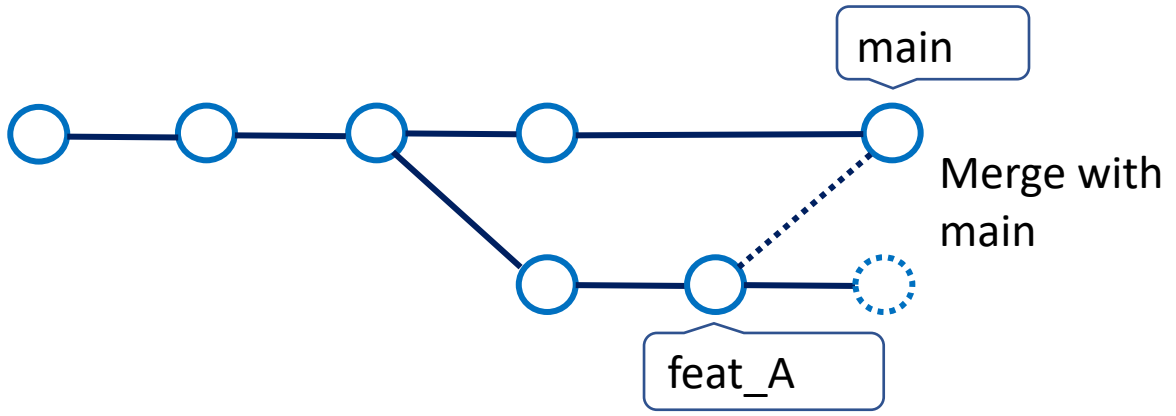
Branching

- When feat_A is ready, it can be merged with main



Branching

- When feat_A is ready, it can be merged with main
- feat_A might be deleted or continue as a persistent branch



Create branch

`git branch [<branch>]`

- List existing branches
  - * = you are here
- Create branch *ext_doc*

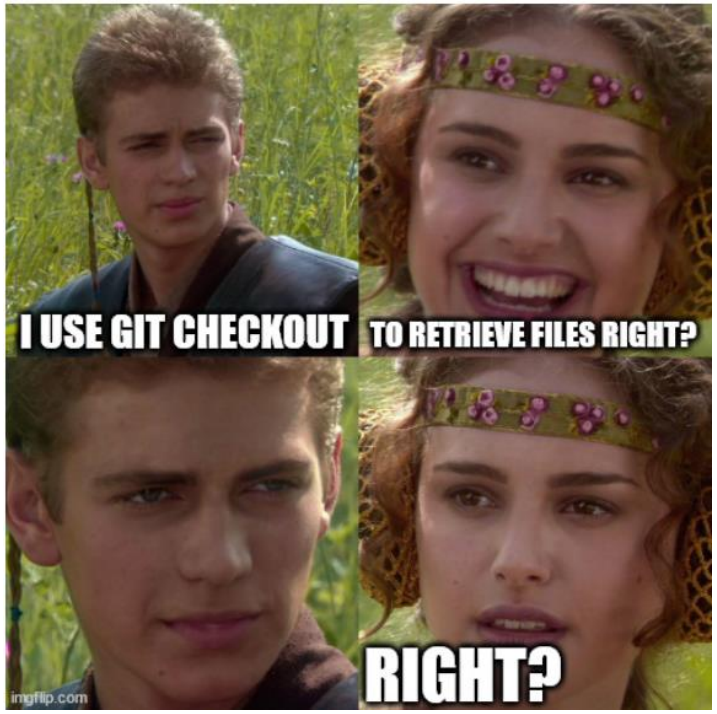
```
$ git branch
* main


$ git branch ext_doc
$ git branch
  ext_doc
* main


$
```

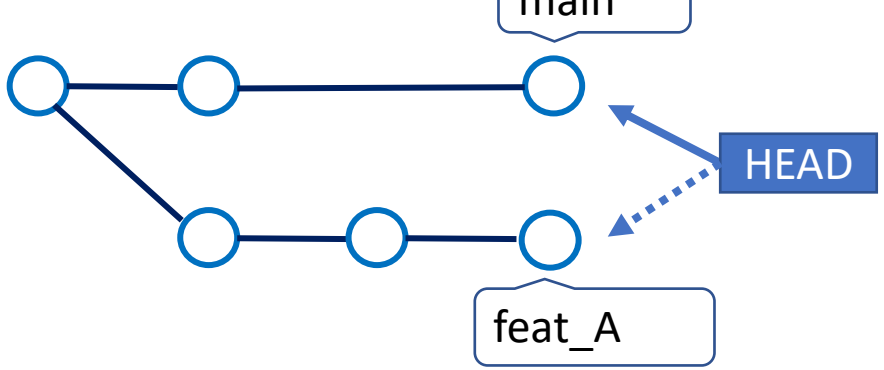
Switch branch

`git checkout <branch>`

Moves HEAD pointer to <branch>

The files in the working area will be changed accordingly.



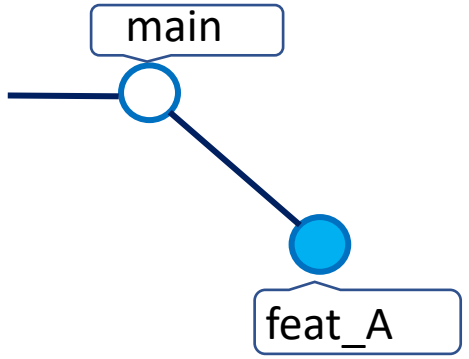



```
$ git branch
* main

$ git branch ext_doc
$ git branch
  ext_doc
* main

$ git checkout ext_doc
M       README.md
Switched to branch 'ext_doc'
$ git branch
* ext_doc
  main
```

Modify branch

- Create a new commit in our branch ext_doc
- Show differences between branches

```
git diff <branch1>..<branch2>
```

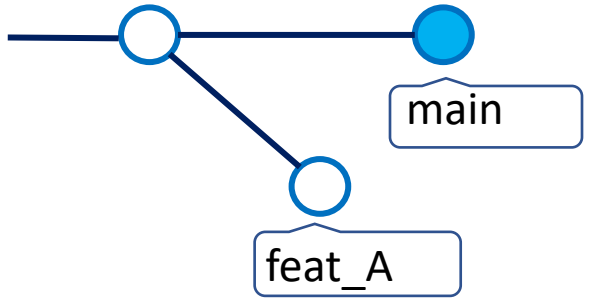



```
$ git commit
[ext_doc 9a546c8] Added documentation
 1 file changed, 5 insertions(+), 1 deletion(-)


$ git diff main..ext_doc
diff --git a/README.md b/README.md
index 4d01659..26949a2 100644
--- a/README.md
+++ b/README.md
@@ -1,3 +1,7 @@
 # test echo

-ver n.
+ver. n+n
+
+Adding more documentation here
+
+bla bla bla...
$
```

Modify main too…

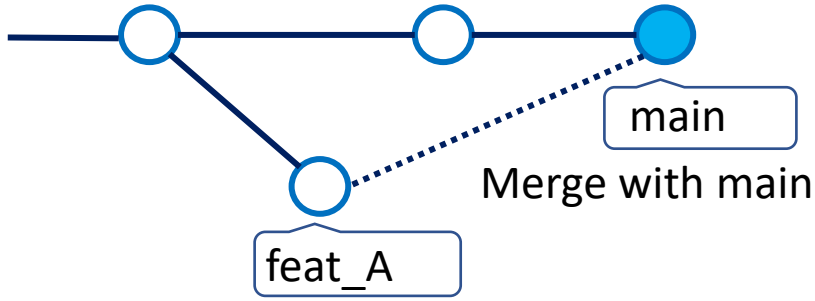
- Switch back to main
- Modify, add and commit
- Show differences between specific file/directory <path>

```
git diff <branch1>..<branch2> <path>
```




```
$ git checkout main
Switched to branch 'main'

$ git add README.md
$ git commit
[main 817e918] Modified title README.md
 1 file changed, 1 insertion(+), 1 deletion(-)

$ git diff main..ext_doc README.md
diff --git a/README.md b/README.md
index 34d698d..26949a2 100644
--- a/README.md
+++ b/README.md
@@ -1,3 +1,7 @@
-# test echooooooooo
+# test echo

-ver n.
+ver. n+n
+
+Adding more documentation here
+
+bla bla bla...
```

Merge branches



- Switch back to main
- Merges changes:

```
git merge <branch>
```

  - This will replay changes made on <branch> since the split, **on the present branch**

Git can merge automatically when changes are:
- in separate files
- on separate locations of the same file

```
$ git checkout main
Switched to branch 'main'


$ git merge ext_doc
Auto-merging README.md
Merge made by the 'recursive' strategy.
 README.md | 6 +++++-
 1 file changed, 5 insertions(+), 1 deletion(-)

$ cat README.md
# test echooooooooo

ver. n+n

Adding more documentation here

bla bla bla...

$
```

Conflict resolution

When merge doesn't come easy…

- If there are 2 conflicting changes on the same line?

After the failed merge

```
#!/bin/bash
echo "test script"
echo "echo more"
echo "echo more and more"
<<<<<<< HEAD
echo "echoo"
=======

echo "echoooo"
>>>>>>> ext_doc
```

Edit to this

```
#!/bin/bash
echo "test script"
echo "echo more"
echo "echo more and more"

echo "echoooo"
```

- Once resolved, add and commit

```
$ git checkout main
Switched to branch 'main'

$ git merge ext_doc
Auto-merging test_echo.sh
CONFLICT (content): Merge conflict in
test_echo.sh
Automatic merge failed; fix conflicts and then
commit the result.

$ git status
On branch main
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   test_echo.sh

no changes added to commit (use "git add"
and/or "git commit -a")

$ git add test_echo.sh
$ git commit
```

Summary

Working with branches	
`git branch`	List local branches (“*” means you are here)
`git branch <branch>`	Create new Branch
`git checkout <branch>`	Change to <branch-name> (updates local directory)
`git diff <branch1>..<branch2> [<path>]`	Check difference between two branches
`git merge <branch>`	Merge changes from <branch-name> into current branch

Exercise (15 mins):

throw a spanner in the works challenge

1. On a repository with 2 files create a new branch

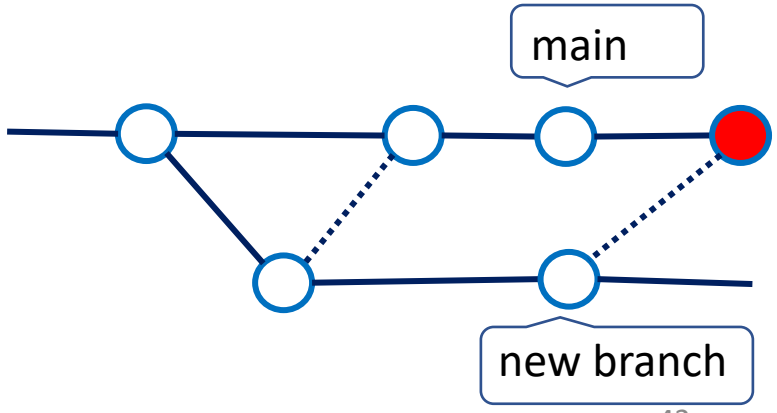
Simple merge

2. On file A, create a commit on only the new branch; checkout main and merge new to main

Break the automatic merge

3. Create commits in file B in both branches and try to cause a merge conflict.

4. Now, sort out the mess...



git stash

- git stops you from checking out when you have local modifications

- If you are not yet ready to commit those modifications, you can *git stash* them away for use later

```
$ git status

On branch old_doc

Changes to be committed:

  (use "git reset HEAD <file>..." to unstage)

        new file:   test_test.sh


Changes not staged for commit:

  (use "git add <file>..." to update what will be
committed)

  (use "git checkout -- <file>..." to discard changes
in working directory)

        modified:   test_echo.sh


$ git checkout main

error: Your local changes to the following files would
be overwritten by checkout:

        test_echo.sh

Please commit your changes or stash them before you
switch branches.

Aborting

$
```

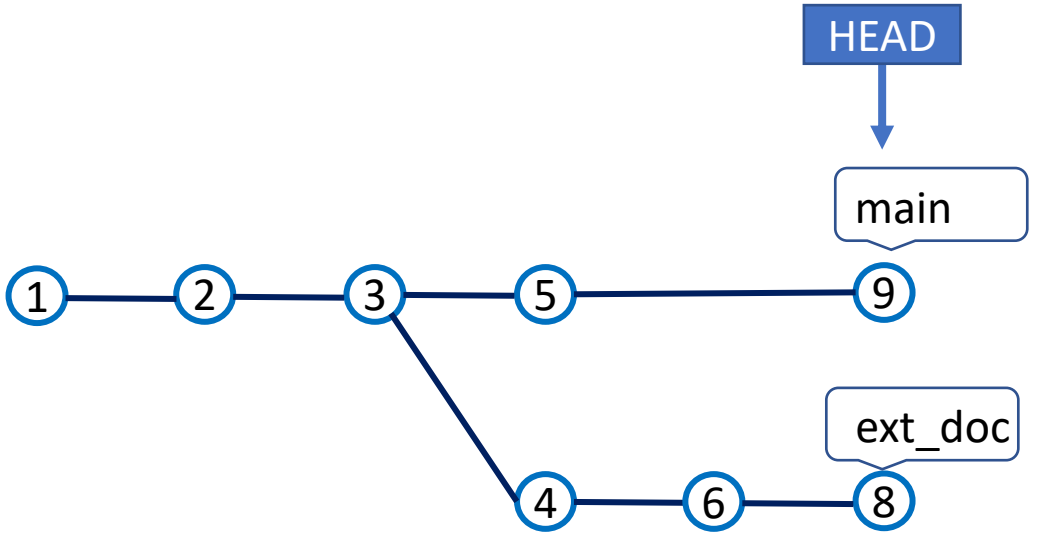
git stash

1. `git stash`
2. `git checkout` to your heart's content
3. `git stash apply` to restore the working directory and staging area

```
$ git stash
Saved working directory and index state WIP on ext_doc: f19
$ git checkout main
Switched to branch 'main'
$ git checkout ext_doc
Switched to branch 'ext_doc'
$ git status
On branch ext_doc
nothing to commit, working tree clean
$ git stash apply
On branch ext_doc
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
        new file:   test_test.sh

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed
  (use "git checkout -- <file>..." to discard changes in wo
        modified:   test_echo.sh

$
```

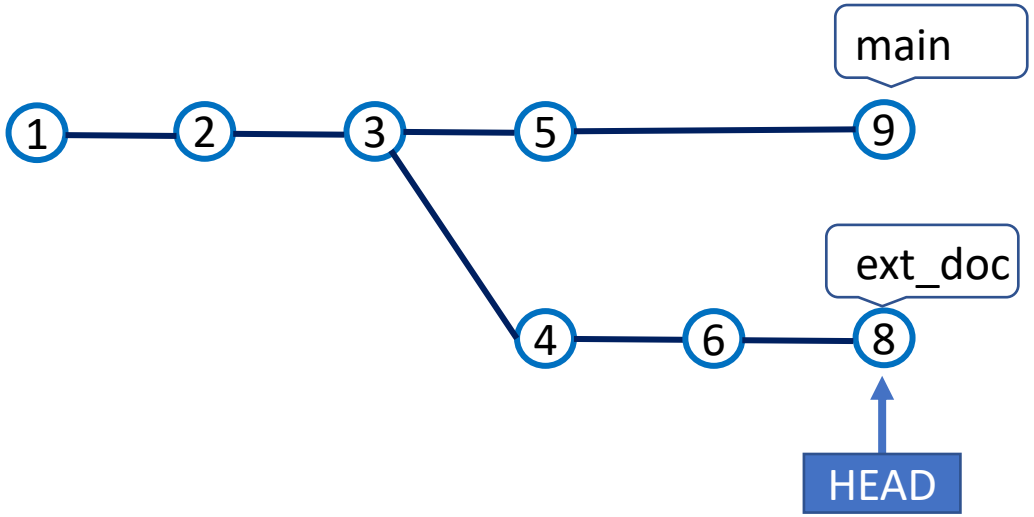
Detached HEAD state and branches

```
$ git checkout main
Switched to branch 'main'
$
```



Detached HEAD state and branches

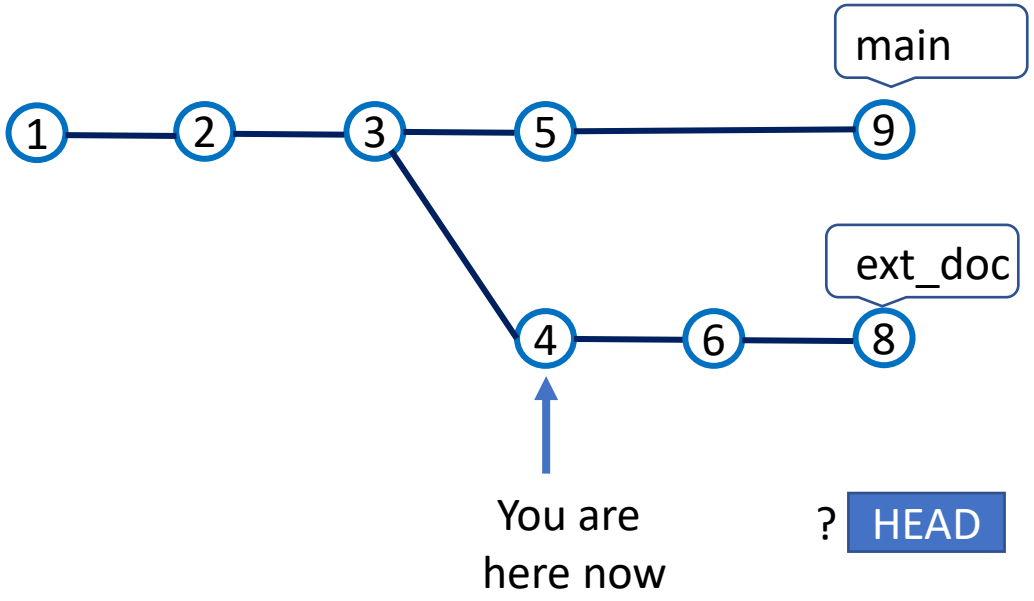
```
$ git checkout main
Switched to branch 'main'
$ git checkout ext_doc
Switched to branch 'ext_doc'
$
```



Detached HEAD state and branches

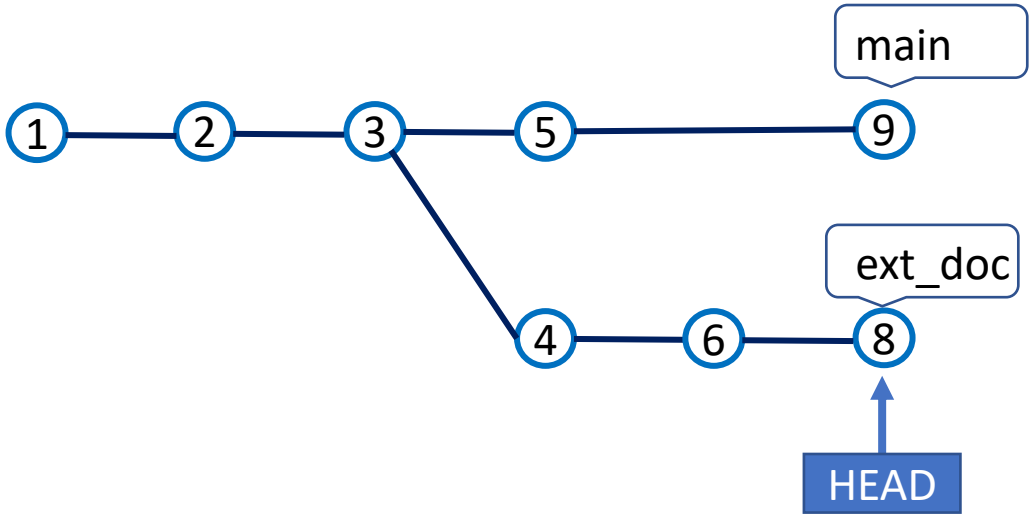
```
$ git checkout 37fdafddc299ab6d4e8ccf3400c6b2b82acdfb6c
Note: checking out '37fdafddc299ab6d4e8ccf3400c6b2b82acdfb6c'.


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 performing another checkout.
(...)
HEAD is now at 37fdafd Revert "Rubbish commit"
$
```

This is a normal state if you are inspecting past commits, but don't make changes as these can be easily lost.



Detached HEAD state and branches

```
$ git checkout 37fdafddc299ab6d4e8ccf3400c6b2b82acdfb6c
Note: checking out '37fdafddc299ab6d4e8ccf3400c6b2b82acdfb6c'.


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 performing another checkout.
(...)
HEAD is now at 37fdafd Revert "Rubbish commit"
$ git checkout ext_doc
Previous HEAD position was 37fdafd Revert "Rubbish commit"
Switched to branch 'ext_doc'
$
```



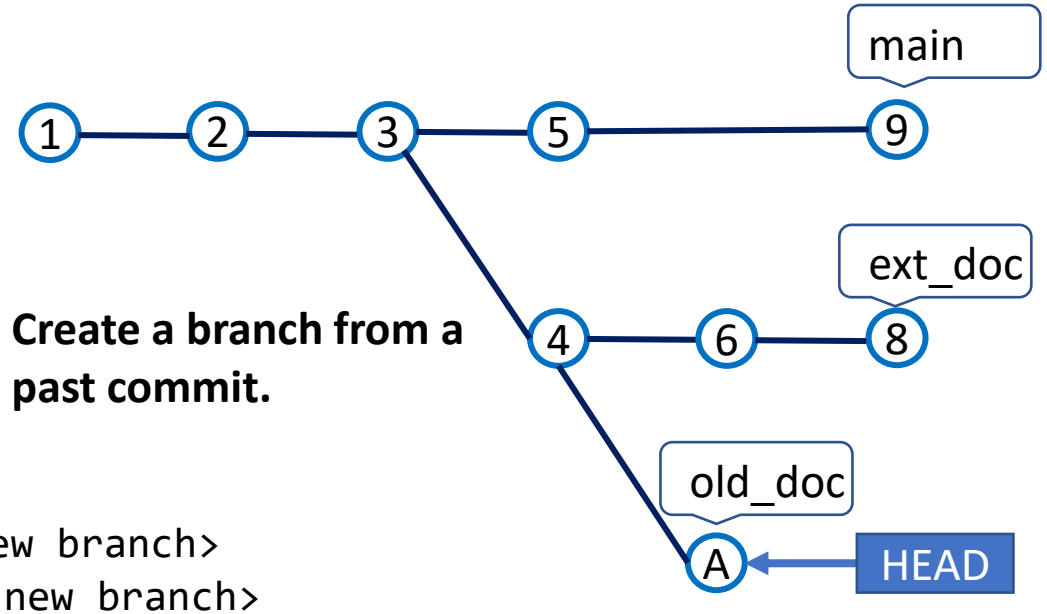
Go back to the end of a branch using

```
git checkout <branch>
```

But what if you want to test changes?

Detached HEAD state and branches

```
$ git checkout 37fdafddc299ab6d4e8ccf3400c6b2b82acdfb6c
(...)

$ git checkout -b old_doc
Switched to a new branch 'old_doc'

(edit file …)

$ git commit
[old_doc 40b7250] added new blank line
 1 file changed, 1 insertion(+)
$
```



Create a branch from a past commit.

`git checkout -b <new branch>` is a shortcut for: `git branch <new branch>`
`git checkout <new_branch>`