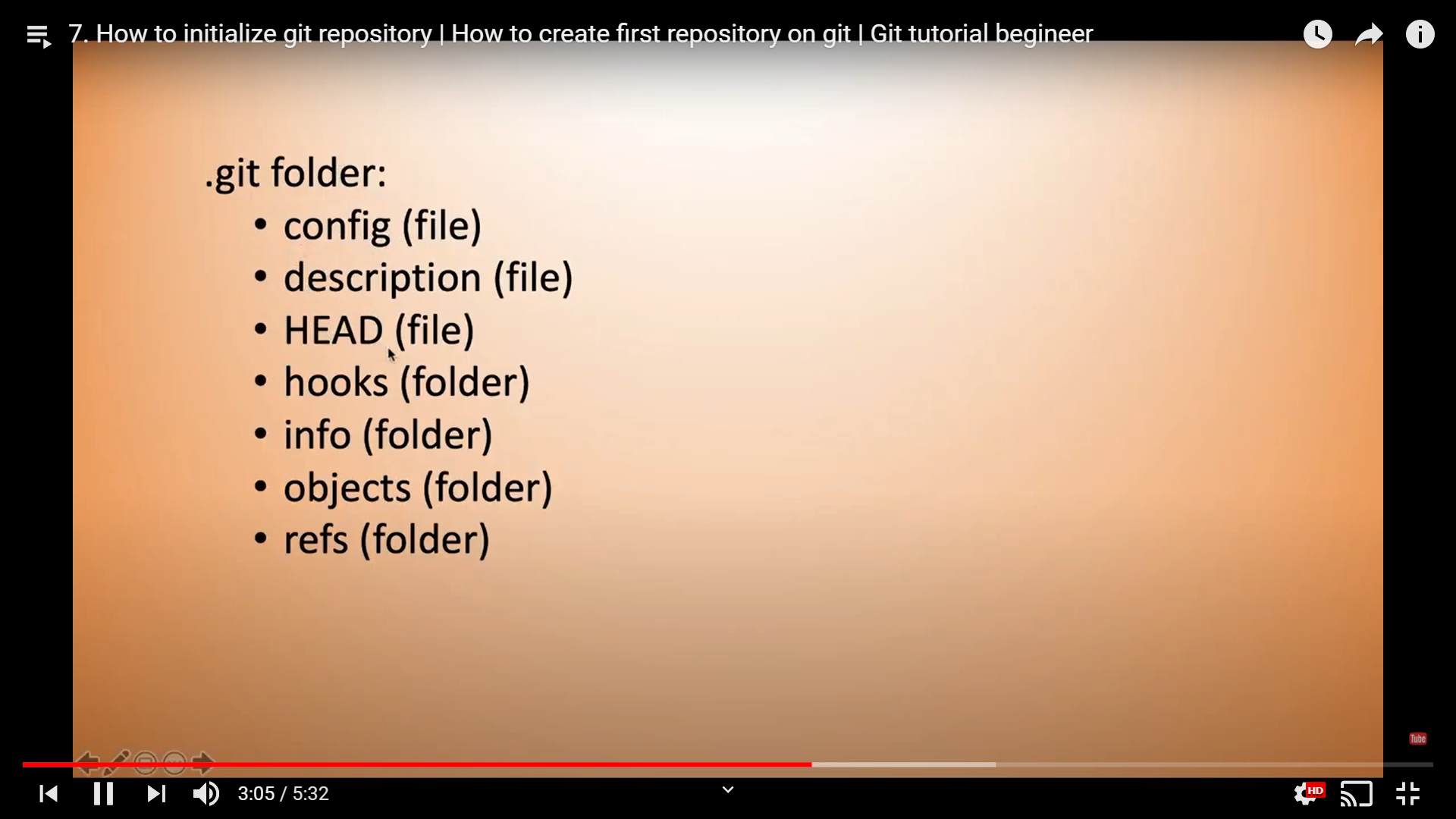


Point to consider before fast forward merge:

1. Extra branch should be created from where we will try to merge the code to receiving branch.
2. Immediate parent of the first commit on new branch should be the last parent of the receiving branch.

End Solution: Master branch pointer will be changed from the last commit of the receiving branch to last commit of new branch.



Here, option --no-ff (i.e. [true merge](https://git-scm.com/docs/git-merge#_true_merge)) creates a new commit with multiple parents, and provides a better history tracking. Otherwise, --ff (i.e. [fast-forward merge](https://git-scm.com/docs/git-merge#_fast_forward_merge)) is by default.

$ git checkout master

$ git checkout -b newFeature

$ ...

$ git commit -m 'work from day 1'

$ ...

$ git commit -m 'work from day 2'

$ ...

$ git commit -m 'finish the feature'

$ git checkout master

$ git merge --no-ff newFeature -m 'add new feature'

$ git log

// something like below

commit 'add new feature' // => commit created at merge with proper message

commit 'finish the feature'

commit 'work from day 2'

commit 'work from day 1'

$ gitk // => see details with graph

$ git checkout -b anotherFeature // => create a new branch (\*)

$ ...

$ git commit -m 'work from day 3'

$ ...

$ git commit -m 'work from day 4'

$ ...

$ git commit -m 'finish another feature'

$ git checkout master

$ git merge anotherFeature // --ff is by default, message will be ignored

$ git log

// something like below

commit 'work from day 4'

commit 'work from day 3'

commit 'add new feature'

commit 'finish the feature'

commit ...

$ gitk // => see details with graph

(\*) Note that here if the newFeature branch is re-used, instead of creating a new branch, git will have to do a --no-ff merge anyway. This means fast forward merge is not always eligible.

There is --no-ff. You can configure merge options per branch, e.g.

git config branch.master.mergeoptions "--no-ff"

adds the following to your $(REPO)/.git/config file:

[branch "master"]

mergeoptions = --no-ff

**Fast Forward Merge**  
  
A **fast**-**forward merge** can occur when there is a linear path from the current branch tip to the target branch. Instead of “actually” **merging** the branches, all Git has to do to integrate the histories is move (i.e., “**fast forward**”) the current branch tip up to the target branch tip.

The –**no**-ff flag causes the **merge** to **always create** a new **commit** object, even if the **merge** could be performed with a **fast**-**forward**. This avoids losing information about the historical existence of a feature branch and groups together all **commits** that together added the feature.

The --**no**-**ff** flag prevents **git merge** from executing a "**fast**-**forward**" if it detects that your current HEAD is an ancestor of the commit you're trying to **merge**. ... A **fast**-**forward** is when, instead of constructing a **merge** commit, **git** just moves your branch pointer to point at the incoming commit.

Merge Conflict

Create a file on Master branch

Git add .

Git commit –m

Create 2nd branch

Edit that file

Git add .

Git commit –m

Again go to master branch

Edit same file

Git merge 2nd branch

Conflict will come

Resolve it

After that , git add . , git commit –m

<https://www.atlassian.com/git/tutorials/using-branches/merge-conflicts>

What does git clean do?

Cleans the working tree by recursively removing files that are not under version control, starting from the current directory. Normally, only files unknown to **Git** are removed, but if the -x option is specified, ignored files are also removed. This can, for example, be useful to remove all build products.

**git clean**

1. If you just **clean** untracked files, **run git clean** -f.
2. If you want to also remove directories, **run git clean** -f -d.
3. If you just want to remove ignored files, **run git clean** -f -X.
4. If you want to remove ignored as well as non-ignored files, **run git clean** -f -x