Version Control & You

By Taro Omiya

What is Version Control?

Version Control, or Source Control Management (SCM) is a set of collaborative tools that allows a team to retain the history of changes in a project

Uh....what does that mean?

OK, a Metaphor

Imagine, you have a time machine.

This time machine requires saving a specific point in time.

Then it gives you the ability to go back to that point in time.

Basically, version controls are elaborate backup solutions.

Introducing Git

- One of the most popular version control from the creator of Linux.
- Commonly used in professional and open-source settings.

Introducing Git

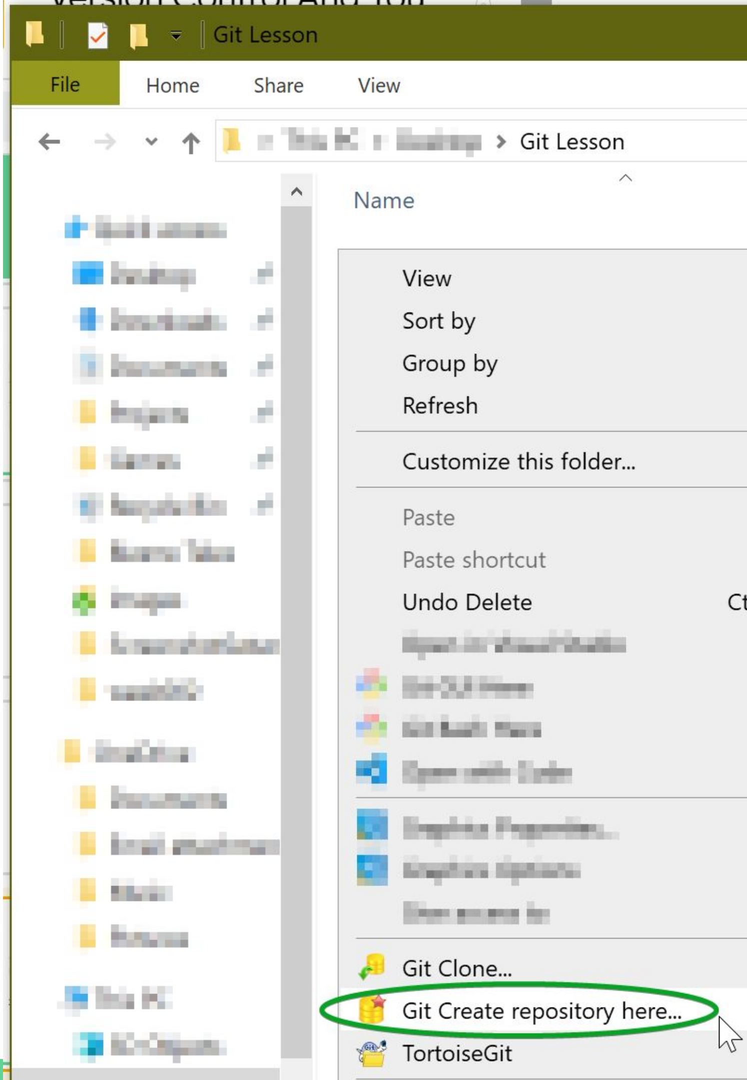
- Git, like most version controls, is a command line tool
- Can be used both online and offline
- This tutorial will use TortoiseGit

The Basics

Creating a Repository

A repository is an encrypted backup of all "versioned" files in a project. For Git, it's held in the `.git` folder.

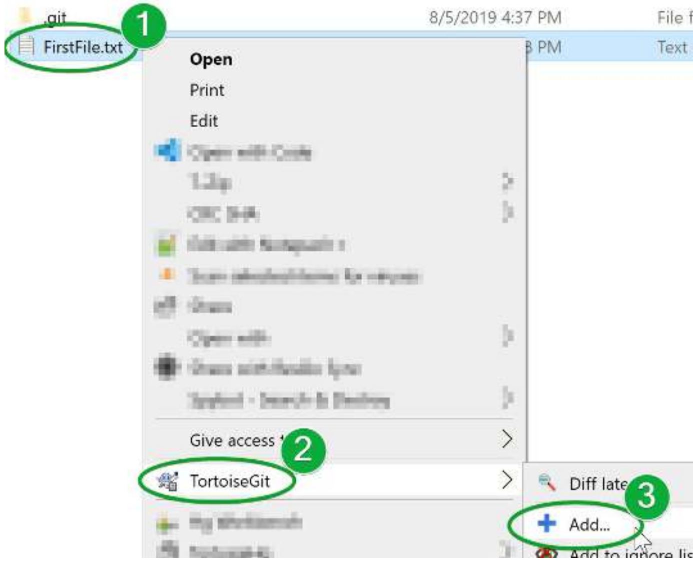
1. Create a new, empty folder, and navigate into it.
2. Right-click the middle of the file explorer.
3. Select "Git Create repository here..."
   a. `git init <directory>`



Adding a File

One must explicitly let Git know which files to version, i.e. to backup:

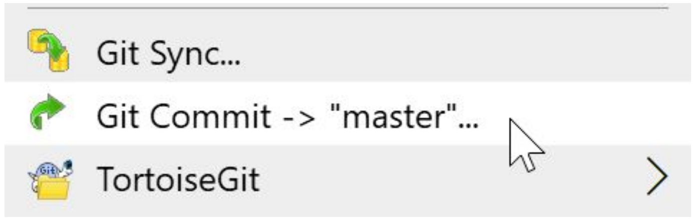
1. Create a new text file, `FirstFile.txt`.
2. Open the file in a text editor (e.g. Notepad), and add the text, "`First line!`"
3. In file explorer, right-click the file and select, "TortoiseGit -> Add..."
   a. `git add <filename>`

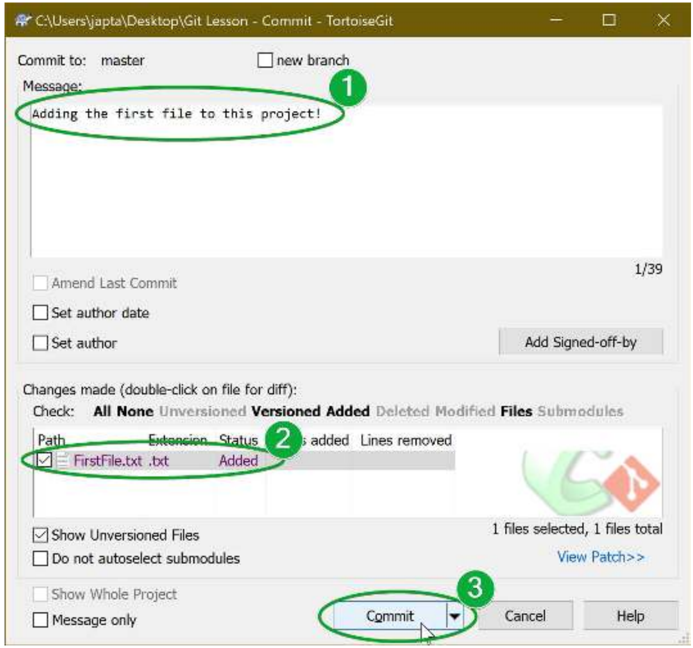


Committing

Finally, a revision (i.e. a save point) needs to be created:

1. Right-click an empty part in the explorer.
2. Select, ‘Git Commit -> “master”…’
   a. `git commit`
3. Write a message at the top of the commit dialog, and confirm `FirstFile.txt` is checked.
4. Click, “Commit”.





C:\Users\japta\Desktop\Git Lesson - Commit - TortoiseGit

Commit to: master ☐ new branch

Message:

```
Adding the first file to this project!
```

1/39

☐ Amend Last Commit
☐ Set author date
☐ Set author

Add Signed-off-by

Changes made (double-click on file for diff):

Check: **All None** Unversioned **Versioned Added** Deleted Modified **Files** Submodules

Path	Extension	Status	s added	Lines removed
☑ FirstFile.txt	.txt	Added		

1 files selected, 1 files total

☑ Show Unversioned Files
☐ Do not autoselect submodules

View Patch>>

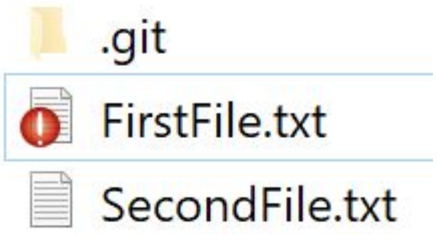
☐ Show Whole Project
☐ Message only

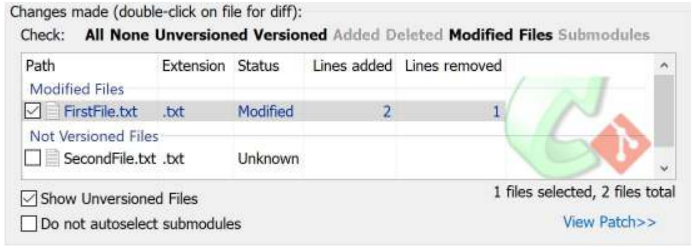
Commit | Cancel | Help

Modifications

Note how TortoiseGit displays modifications:

1. Open `FirstFile.txt` in a text editor.
2. Add a new line, “`Second Line!`”
3. Create a new text file, `SecondFile.txt`
   a. This file may remain empty.
4. Right-click an empty part in the explorer.
5. Select, ‘Git Commit -> “master”...’




Changes made (double-click on file for diff):

Check: **All** **None** **Unversioned** **Versioned** Added Deleted **Modified** **Files** Submodules

Path	Extension	Status	Lines added	Lines removed
Modified Files				
☑ FirstFile.txt	.txt	Modified	2	1
Not Versioned Files				
☐ SecondFile.txt	.txt	Unknown		

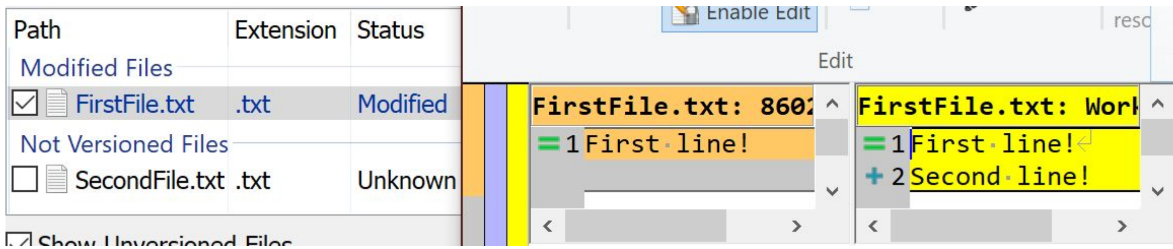
☑ Show Unversioned Files

☐ Do not autoselect submodules

1 files selected, 2 files total

View Patch>>

Modifications



Path	Extension	Status
Modified Files		
✓ FirstFile.txt	.txt	Modified
Not Versioned Files		
☐ SecondFile.txt	.txt	Unknown

```
FirstFile.txt: 860
1 First·line!
```

```
FirstFile.txt: Work
1 First·line!
2 Second·line!
```

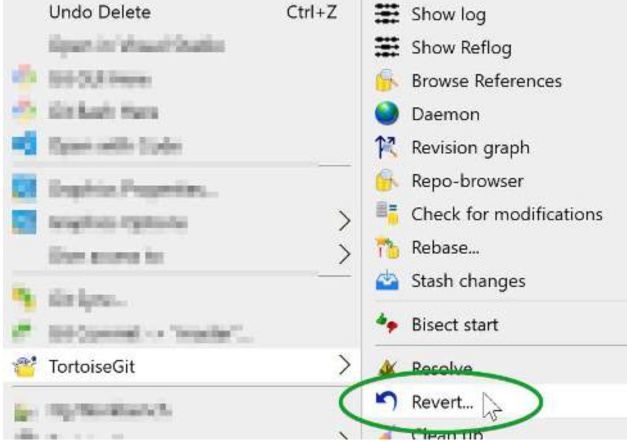
Note how TortoistGit displays modifications:

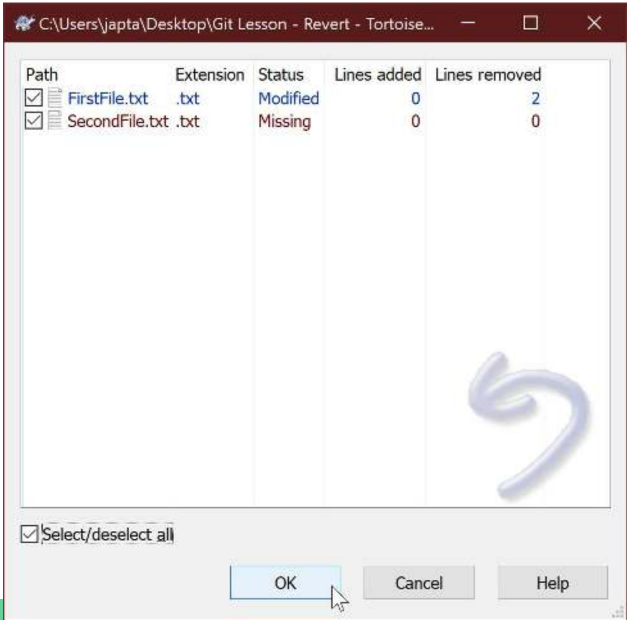
1. In the commit dialog, double-click `FirstFile.txt`
2. The new dialog provides highlights on what lines have changed.
3. Close this dialog. Also note “`SecondFile.txt`” is not checked.
4. Enter a message, then check `SecondFile.txt`, and finally click, Commit.
    a. Checking an unversioned file in a commit dialog also adds the file to the repository.

Reverting Files

Git can easily revert or reset a project to the latest state stored in the repository:

1. Open `FirstFile.txt` in a text editor, then delete its content and save.
2. Delete `SecondFile.txt`
3. Right-click an empty part in the explorer.
4. Select, “TortoiseGit -> Revert…”
   a. `git reset`
5. Check all files in the revert dialog, and confirm.



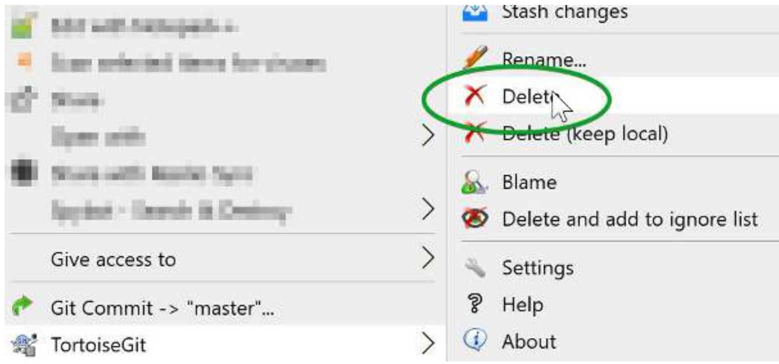


Path	Extension	Status	Lines added	Lines removed
☑ FirstFile.txt	.txt	Modified	0	2
☑ SecondFile.txt	.txt	Missing	0	0

Deleting a File

A file must be marked for removal so the repository knows to stop tracking it as well:

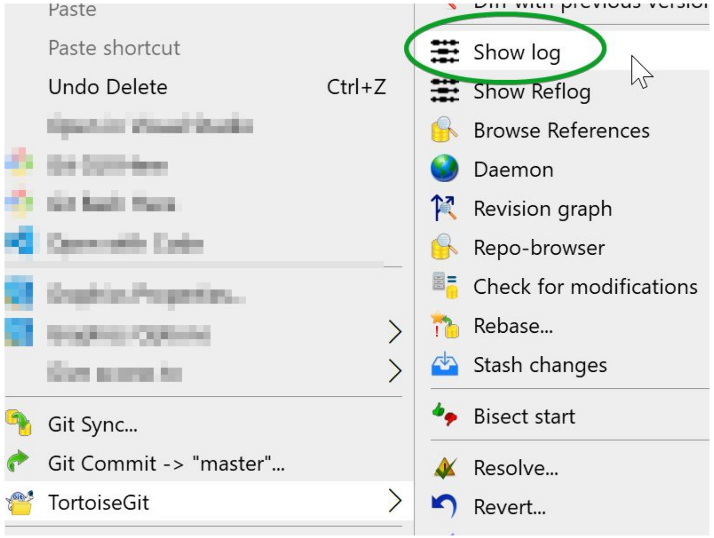
1. Right-click `SecondFile.txt`
2. Select, “TortoiseGit -> Delete”
   a. `git rm`
3. Commit changes.



Displaying the Repository History

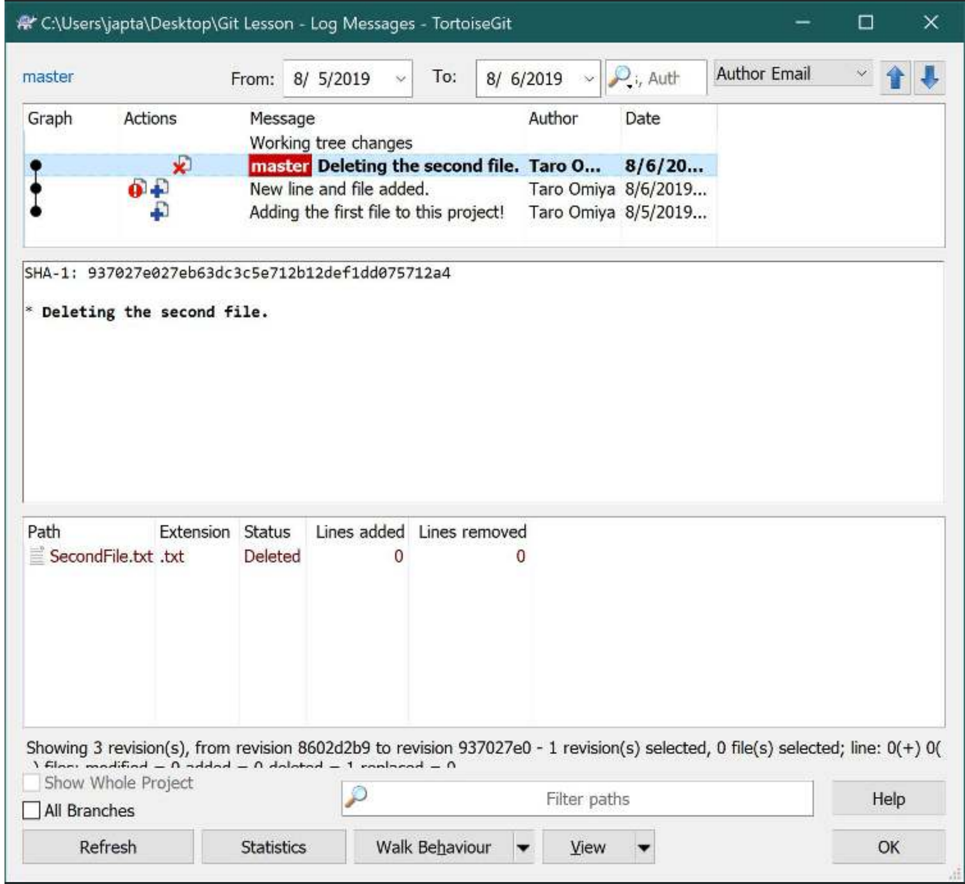
To show all the changes made in the repository so far, use “log:”

1. Right-click an empty part in the explorer.
2. Select, “TortoiseGit -> Show log”



The Log dialog not only shows past messages at the top and middle of the dialog, but also what files has changed at the bottom of the dialog.

Not only that, but one can also review *how* each file changes by simply double-clicking them.



From: 8/ 5/2019 To: 8/ 6/2019

Graph	Actions	Message	Author	Date
		Working tree changes		
●		**master Deleting the second file.**	**Taro O...**	**8/6/20...**
●		New line and file added.	Taro Omiya	8/6/2019...
●		Adding the first file to this project!	Taro Omiya	8/5/2019...

```
SHA-1: 937027e027eb63dc3c5e712b12def1dd075712a4

* Deleting the second file.
```

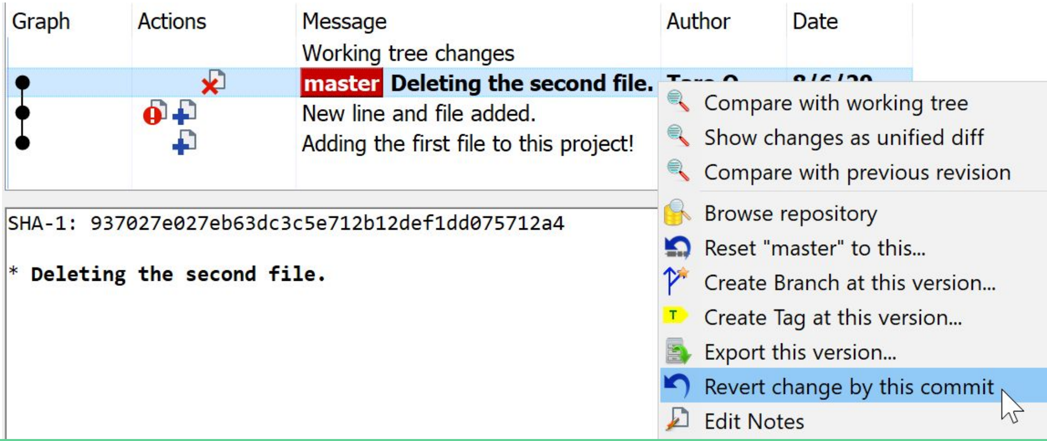
Path	Extension	Status	Lines added	Lines removed
SecondFile.txt	.txt	Deleted	0	0

Showing 3 revision(s), from revision 8602d2b9 to revision 937027e0 - 1 revision(s) selected, 0 file(s) selected; line: 0(+) 0(
) files: modified = 0 added = 0 deleted = 1 replaced = 0

Reverse-Merging a File

How does one revert changes that has already been committed? By reverse-merging:

1. In the Log dialog, right-click the latest revision.
2. Select, “Revert change by this commit”.
   a. `git revert <rev>`
   b. Where `<rev>` is a hash
   c. (see first line of each log)
3. Click OK.
4. Commit changes.



Graph	Actions	Message	Author	Date
		Working tree changes		
●		master Deleting the second file.		
●		New line and file added.		
●		Adding the first file to this project!		

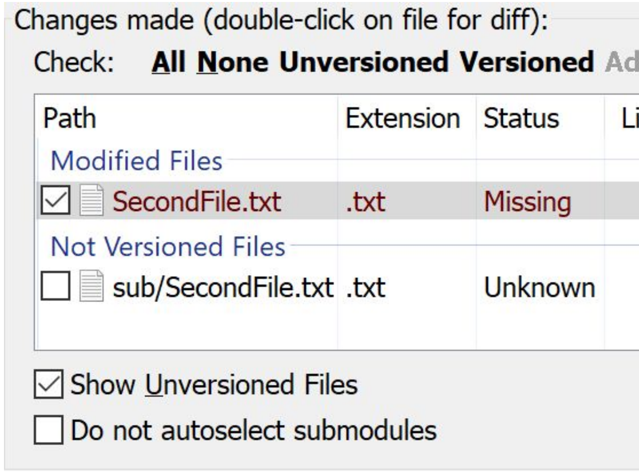
```
SHA-1: 937027e027eb63dc3c5e712b12def1dd075712a4

* Deleting the second file.
```

Compare with working tree
Show changes as unified diff
Compare with previous revision
Browse repository
Reset "master" to this...
Create Branch at this version...
Create Tag at this version...
Export this version...
Revert change by this commit
Edit Notes

Moving a File

1. Create a new folder named, “`sub`”.
2. Move the “`SecondFile.txt`” into “`sub`”.
3. Right-click an empty part in the explorer.
4. Select, ‘Git Commit -> “master”...’
   a. Do *not* commit.
5. Observe changes that were detected:
   a. Technically, Git can version a file moved this way, but it’ll create problems in the future. There is a better way.



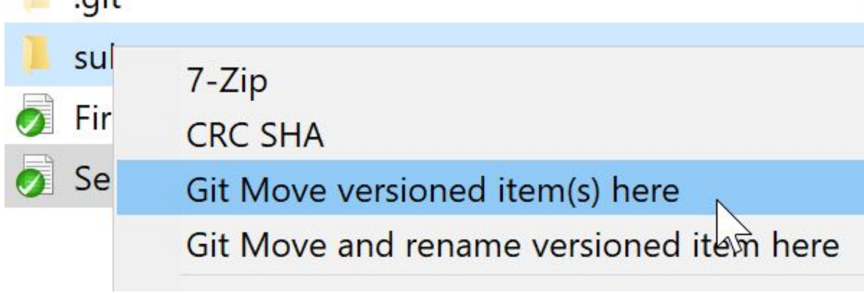
Changes made (double-click on file for diff):

Check: **All None Unversioned Versioned Ad**

Path	Extension	Status	Li
Modified Files			
✓ SecondFile.txt	.txt	Missing	
Not Versioned Files			
☐ sub/SecondFile.txt	.txt	Unknown	

✓ Show Unversioned Files

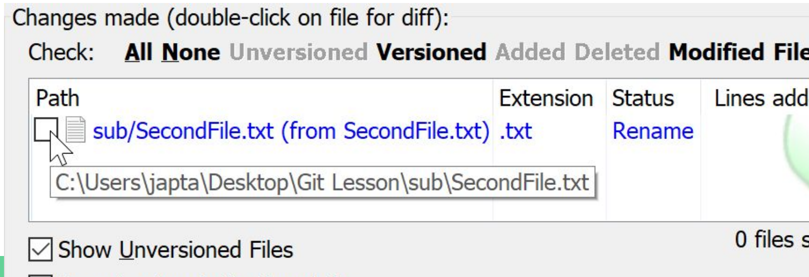
☐ Do not autoselect submodules


.git
sub
Fir
Se
7-Zip
CRC SHA
Git Move versioned item(s) here
Git Move and rename versioned item here

Moving a File

Git has a special way of marking moved or renamed files:

1. Return "`SecondFile.txt`" back to the original folder.
2. Right-click, hold, then drag the file into the "`sub`" folder. Upon letting go, a unique context menu will show up.
3. Select "Git Move versioned item(s) here".
   a. `git mv <old-filename> <new-filename>`
4. Commit changes.
   a. Notice status appears as "Rename"


Changes made (double-click on file for diff):
Check: All None Unversioned Versioned Added Deleted Modified File
Show Unversioned Files
0 files s

Path	Extension	Status	Lines add
sub/SecondFile.txt (from SecondFile.txt)	.txt	Rename	

C:\Users\japta\Desktop\Git Lesson\sub\SecondFile.txt

A Quick Review

- A file needs to be marked by the `add` command, or else git will not backup the file.
- The command, `commit` creates a new revision, a save point in the project's timeline. A revision usually has a custom message attached. These messages can be reviewed under the log.
- `commit` automatically detects modifications to a file, and version them accordingly. Exceptions are:
  a. If a file should be delete, use the `remove` command.
  b. If a file should be moved to another folder, use the `move` command.
  c. If a file should be renamed, (oddly) use the `move` command.

A Quick Review

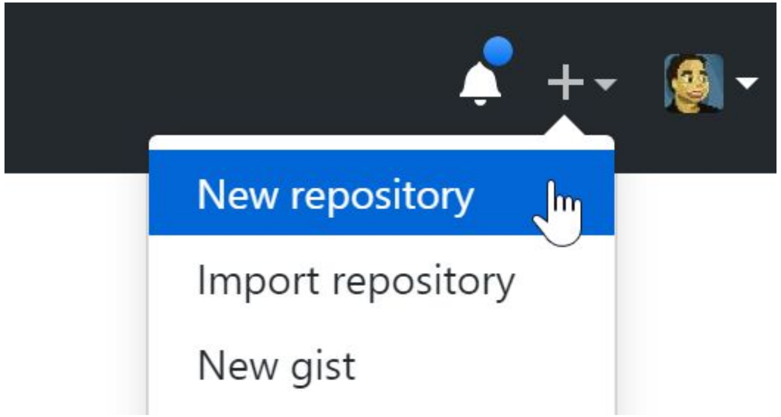
- The `reset` and `revert` command allows one to return the project to an older revision. The former returns the project to it's latest state; the latter reverses a change from an older revision.
- These changes works wonderfully in text files, but not in binary or encrypted files.
  - Code, HTML pages, etc. are easy to revert.
  - Images, sound, music, Word documents, etc. are harder to revert.
    - Not to mention these files inflate the repository size quite significantly.
  - Oddly, even if they're converted to text, Unity scenes and prefabs are hard to revert as well.
    - Still recommended due to keeping the repository size small.

Collaboration

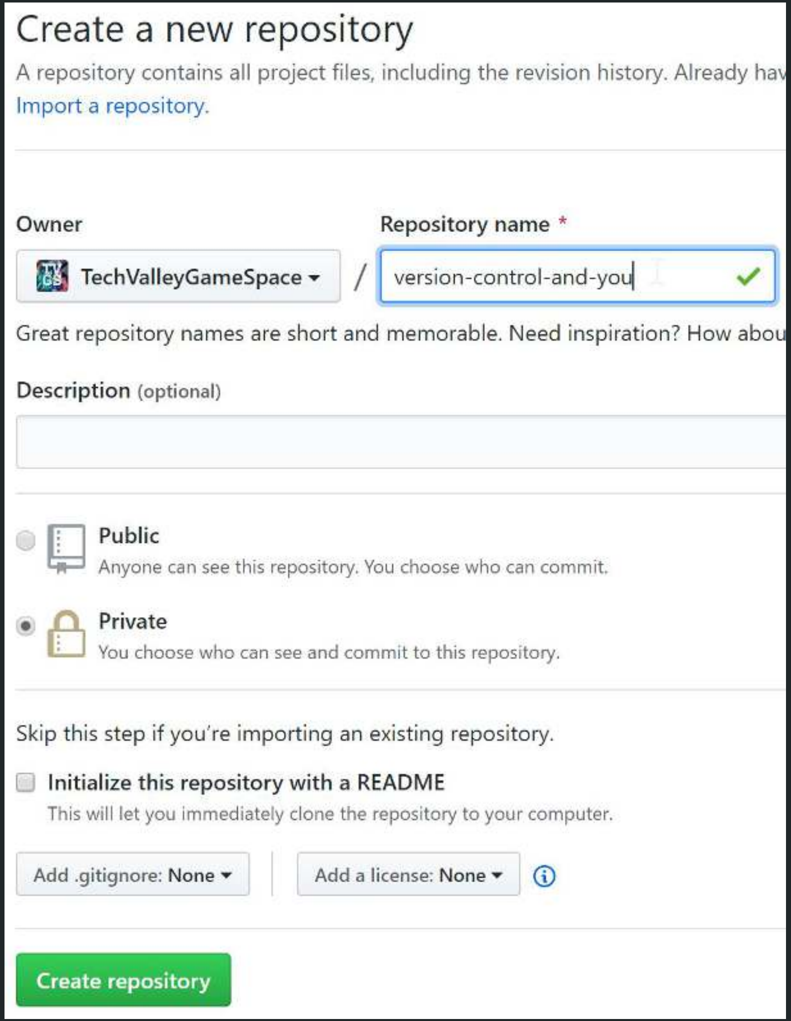
Create an Online Repository

GitHub, Bitbucket, and other online repositories provides one to share their project to other people:

1. Create a new repository on GitHub (or an online repository of choice).



2. Give the repository a name
3. Adjust any settings
    a. For the purpose of this lesson, the repository will be marked public, but feel free to setup a private repository.
4. Since we're importing an existing repository, leave all options at the bottom section unchecked and set to "None."
5. Click, "Create repository."



Create a new repository

A repository contains all project files, including the revision history. Already hav

Import a repository.

Owner: TechValleyGameSpace / Repository name *: version-control-and-you ✓

Great repository names are short and memorable. Need inspiration? How abou

Description (optional)

Public
Anyone can see this repository. You choose who can commit.

[✓ Private]
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

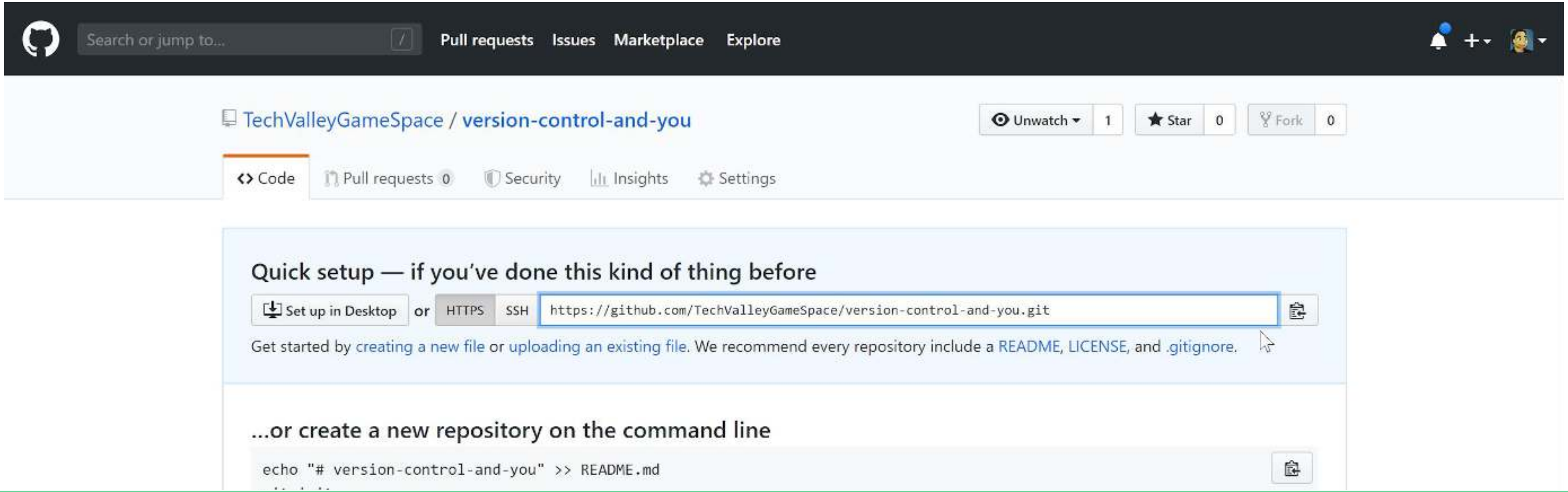
☐ Initialize this repository with a README
This will let you immediately clone the repository to your computer.

Add .gitignore: None | Add a license: None

Create repository

Record the Online Repository's URL

1. Copy the URL at the top of the webpage.
    a. Make sure HTTPS is selected (rather than SSH)



Quick setup — if you've done this kind of thing before

```
https://github.com/TechValleyGameSpace/version-control-and-you.git
```

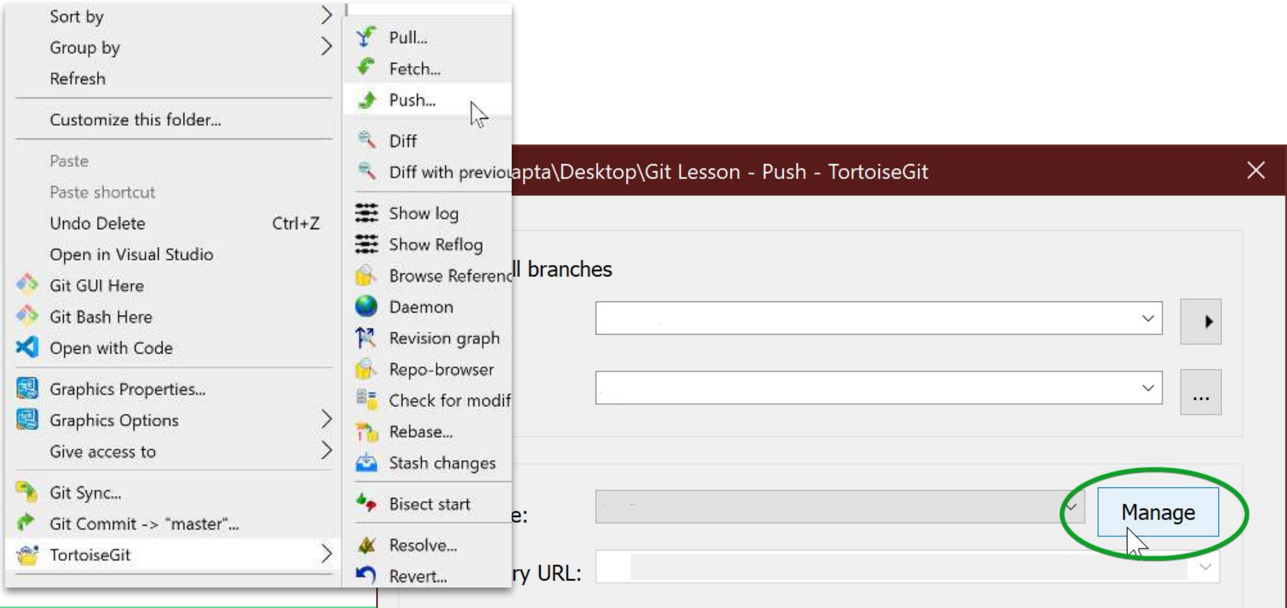
Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

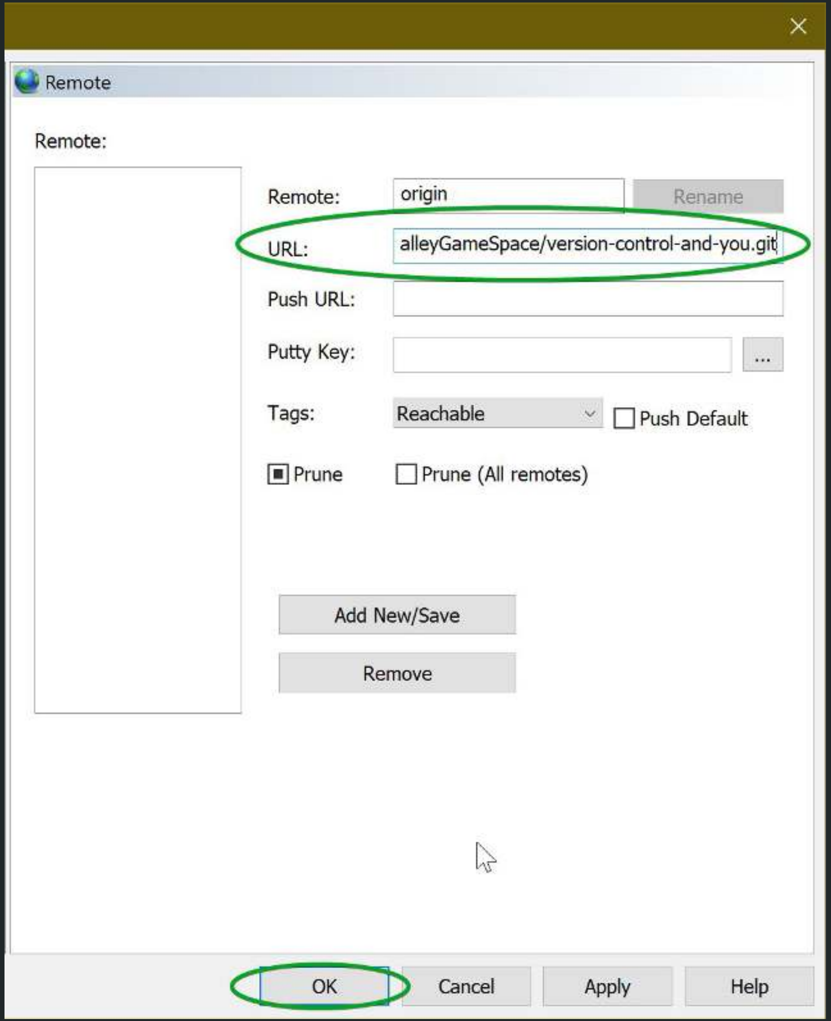
```
echo "# version-control-and-you" >> README.md
```

Record the Online Repository's URL

2. Right-click an empty part in the explorer.
3. Select, "TortoiseGit -> Push..."
4. In the Push dialog,
   Click "Manage"



5. Paste the repo's URL under the "URL" field.
    a. Notice the field, "Remote" will automatically be filled with the default value, "origin." This is normal.
6. Click OK.
    a. `git remote add origin <url>`
    b. Essentially, this dialog creates a bookmark named, "`origin`" that points to the entered URL


Remote

Remote:

Remote: origin

URL: alleyGameSpace/version-control-and-you.git

Push URL:

Putty Key:

Tags: Reachable

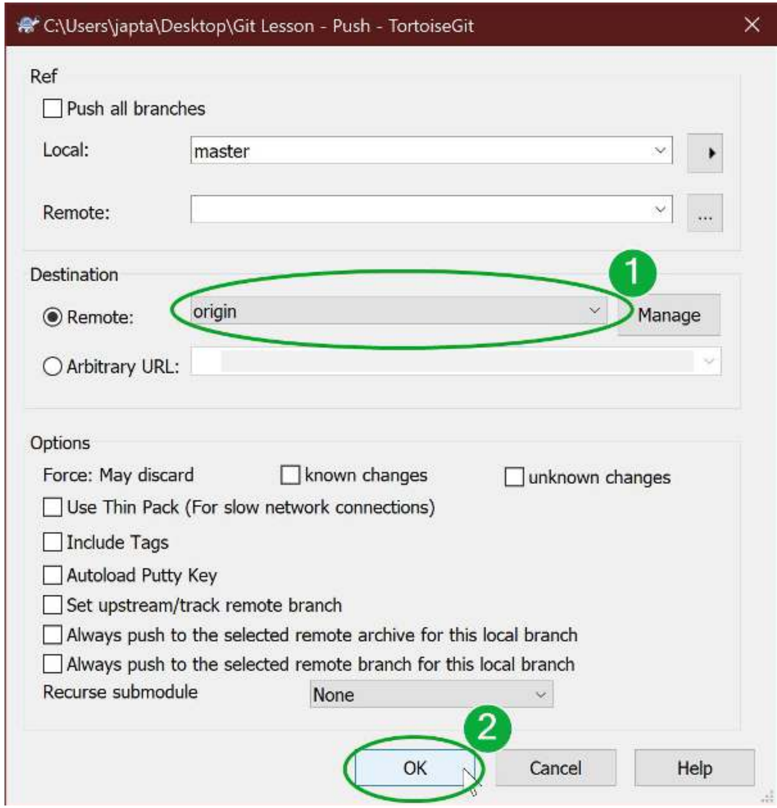
☐ Push Default

☑ Prune

☐ Prune (All remotes)

Upload Project Online

1. Confirm the field, “Remote” is filled with “origin”.
2. Click, “OK”.



C:\Users\japta\Desktop\Git Lesson - Push - TortoiseGit

Ref

- [ ] Push all branches

Local: master

Remote:

Destination

(●) Remote: origin

( ) Arbitrary URL:

Options

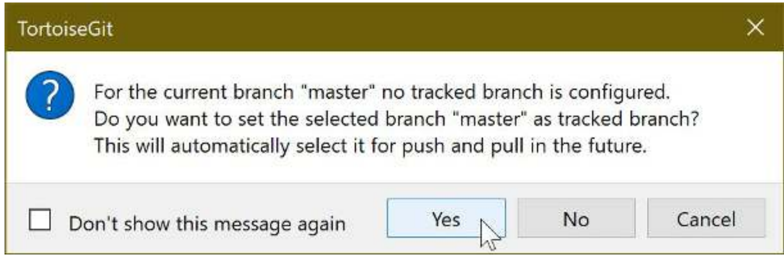
Force: May discard [ ] known changes [ ] unknown changes

- [ ] Use Thin Pack (For slow network connections)
- [ ] Include Tags
- [ ] Autoload Putty Key
- [ ] Set upstream/track remote branch
- [ ] Always push to the selected remote archive for this local branch
- [ ] Always push to the selected remote branch for this local branch

Recurse submodule: None

Upload Project Online

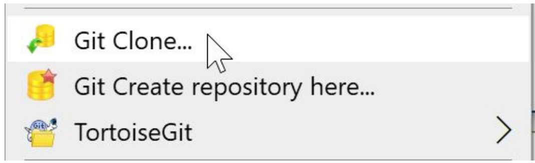
3. On the next pop-up dialog, select “Yes”
    a. `git push -u origin master`
    b. The dialog is asking whether the branch, “`master`” should be created in the remote repository. What “branch” means is a bit beyond the scope of this lesson.
4. Enter your credentials for GitHub.

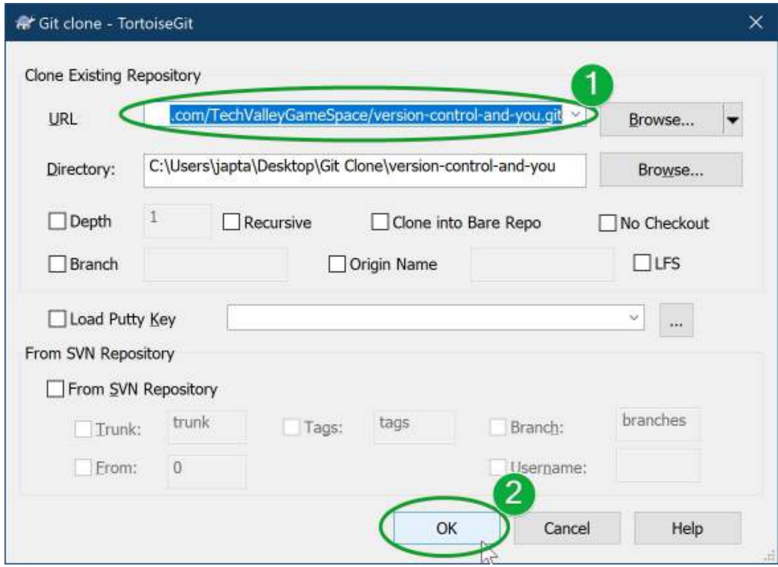

TortoiseGit

For the current branch "master" no tracked branch is configured.
Do you want to set the selected branch "master" as tracked branch?
This will automatically select it for push and pull in the future.

Download Online Project

1. Navigate *out* of the project.
2. Right-click on an empty part of the file explorer, then select “Git Clone...”
3. Enter the same URL as the one copied from the online repository.
   a. `git clone <url>`
4. Click OK.
   a. Note: if the online repository is private, enter your GitHub credentials.




Git clone - TortoiseGit

Clone Existing Repository

1

URL: .com/TechValleyGameSpace/version-control-and-you.git Browse...

Directory: C:\Users\japta\Desktop\Git Clone\version-control-and-you Browse...

☐ Depth 1 ☐ Recursive ☐ Clone into Bare Repo ☐ No Checkout

☐ Branch ☐ Origin Name ☐ LFS

☐ Load Putty Key ...

From SVN Repository

☐ From SVN Repository

☐ Trunk: trunk ☐ Tags: tags ☐ Branch: branches

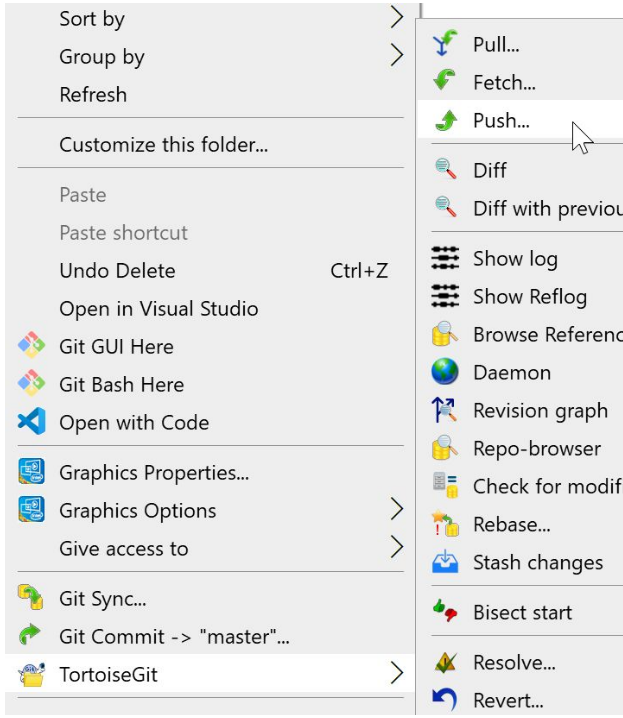
☐ From: 0 ☐ Username:

2

OK Cancel Help

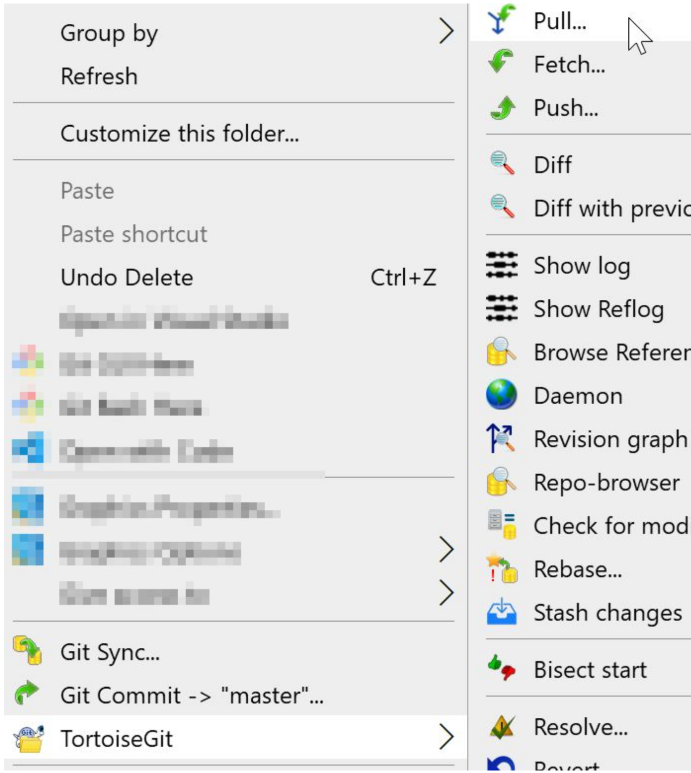
Sharing Changes

1. In the cloned project, change the content of “`sub/SecondFile.txt`” to, “`I was here`”
2. Commit this change.
3. Push this change.
    a. Note: most settings in the Push dialog should be already set; just click, “OK”.
4. Enter GitHub credentials.



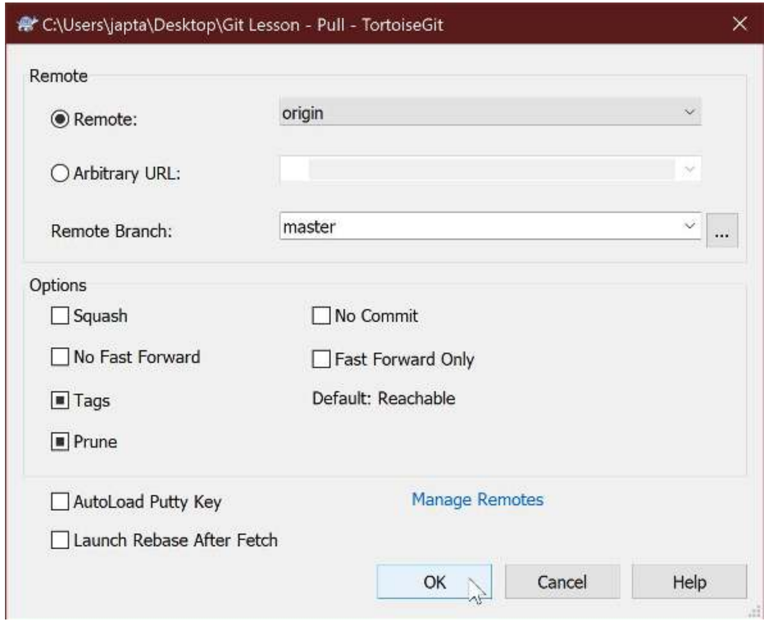
Sharing Changes

1. Navigate to the original project.
2. Right-click on an empty part of the file explorer.
3. Select “TortoiseGit -> Pull...”



Sharing Changes

1. In the Pull dialog, most settings should be properly filled in.
2. Click OK.
3. Enter credentials.
4. Verify `SecondFile.txt` contains new content.



C:\Users\japta\Desktop\Git Lesson - Pull - TortoiseGit

Remote

[● Remote:] origin

○ Arbitrary URL:

Remote Branch: master

Options

☐ Squash
☐ No Commit
☐ No Fast Forward
☐ Fast Forward Only
▣ Tags
Default: Reachable
▣ Prune

☐ AutoLoad Putty Key
Manage Remotes
☐ Launch Rebase After Fetch

OK Cancel Help

A Quick Review

When an online repository is involved, there are now two levels of repositories involved: one on your computer (local), and one online (remote).

- `git commit` adds revisions to your local repository.
- `git push` uploads all the commits in the local repository to the remote one.
- `git pull` is actually two-commands-in-one:
  - `git fetch` downloads commits from the remote repository to the local one.
  - `git merge` applies the latest changes from the local repository to your current project.

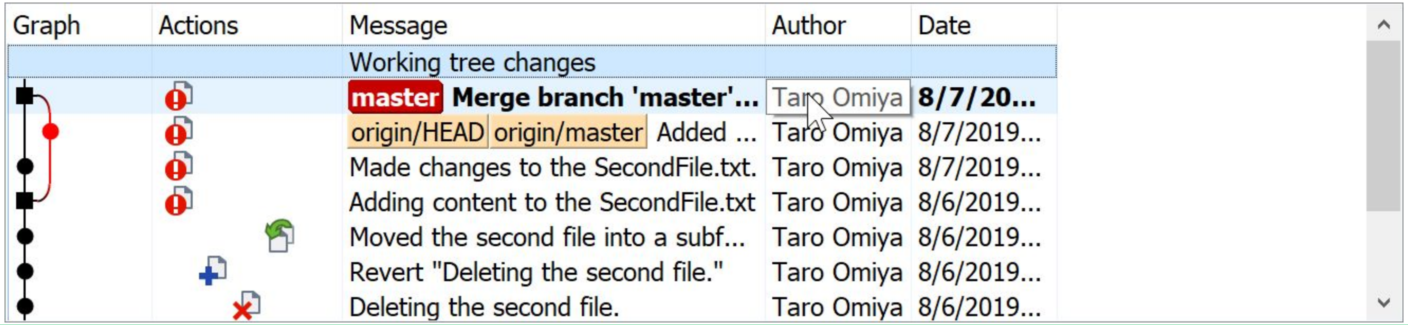
Merging Differences

Auto-Merge

1. In the original project, add yet another line in `FirstFile.txt` with “`Third line is the charm!`”
2. Commit, then push this change.
3. Navigate to the cloned project.
4. Open “`sub/SecondFile.txt`”, then overwrite the content with, “`Changes made from cloned project.`”
5. *Only* commit this change to the cloned project.
6. Pull changes from the remote repository.
7. Verify changes in `FirstFile.txt`.

Auto-Merge

1. While in the cloned project, bring up the log.
2. Note the graph column indicating a merge occured.
   a. In particular, notice on each line which file changed, and where the source of the change came from.



Graph	Actions	Message	Author	Date
		Working tree changes		
		master Merge branch 'master'...	Taro Omiya	**8/7/20...**
		origin/HEAD origin/master Added ...	Taro Omiya	8/7/2019...
		Made changes to the SecondFile.txt.	Taro Omiya	8/7/2019...
		Adding content to the SecondFile.txt	Taro Omiya	8/6/2019...
		Moved the second file into a subf...	Taro Omiya	8/6/2019...
		Revert "Deleting the second file."	Taro Omiya	8/6/2019...
		Deleting the second file.	Taro Omiya	8/6/2019...

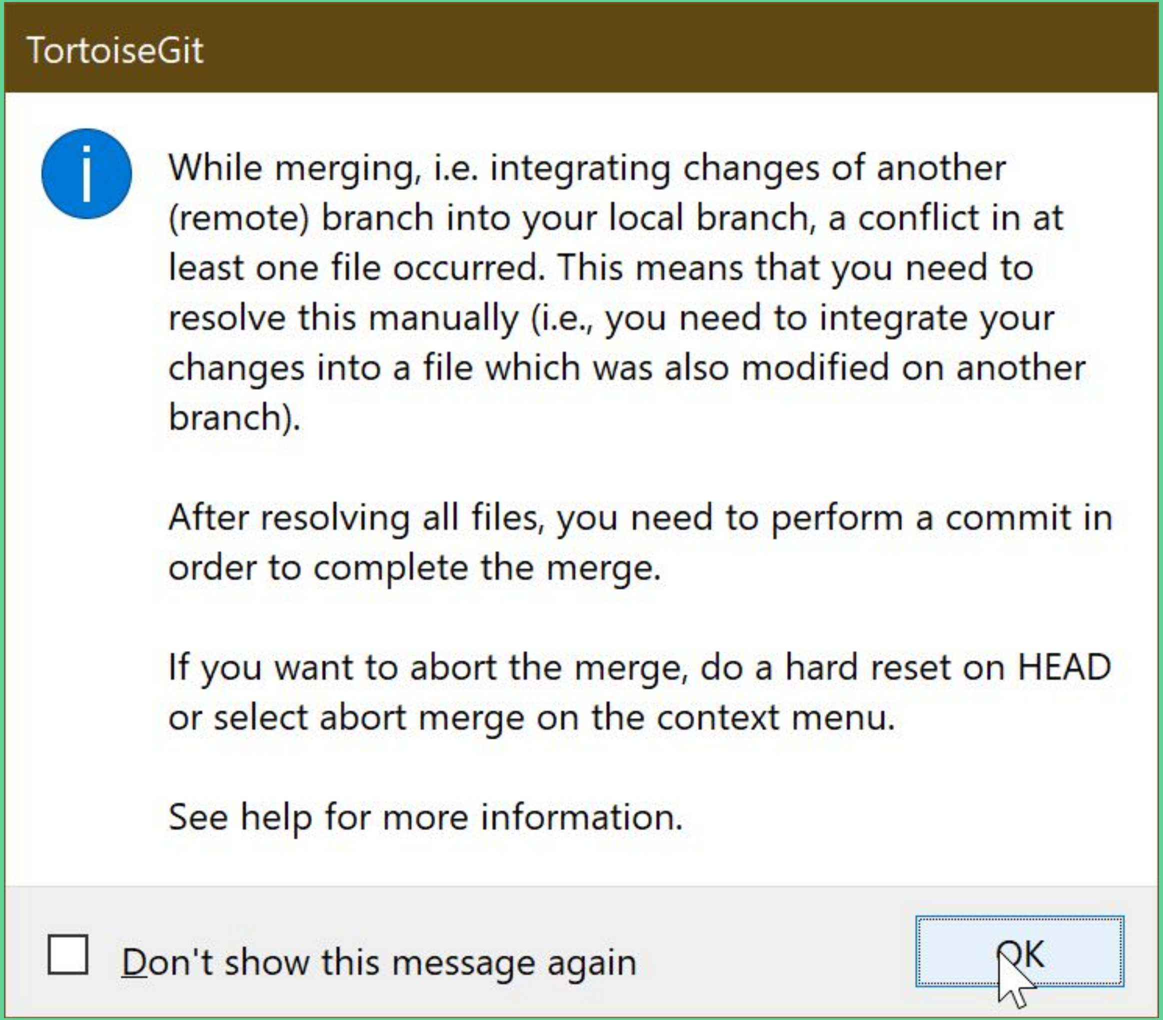
A Quick Review

Git generally tries to automatically merge changes from two or more people as smoothly as possible.

- If each person works on a different file, usually, this goes very smoothly.
- If two or more people make changes on the same file, however, a merge conflict may occur.
- Merge conflict occurs when Git detects two or more people made changes to (approximately) the *same lines* in the same file.
- Important: as a general policy, always commit changes first before pulling!
    - It keeps all changes recorded in the repository.

Simulating a Merge Conflict

1. Remember we made a change to “`sub/SecondFile.txt`” in the cloned project? Let’s push this change to the remote repository.
2. Navigate back to the original project.
3. Overwrite the content of “`sub/SecondFile.txt`” with “`Changes made from original project.`”
4. Commit this change.
5. Pull from the remote server.


TortoiseGit

While merging, i.e. integrating changes of another (remote) branch into your local branch, a conflict in at least one file occurred. This means that you need to resolve this manually (i.e., you need to integrate your changes into a file which was also modified on another branch).

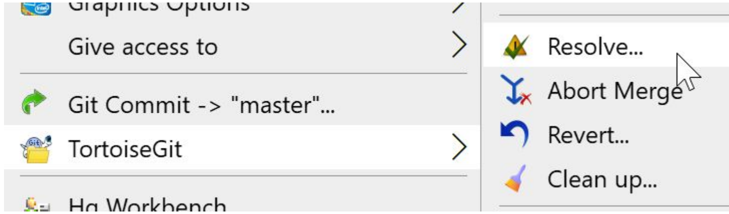
After resolving all files, you need to perform a commit in order to complete the merge.

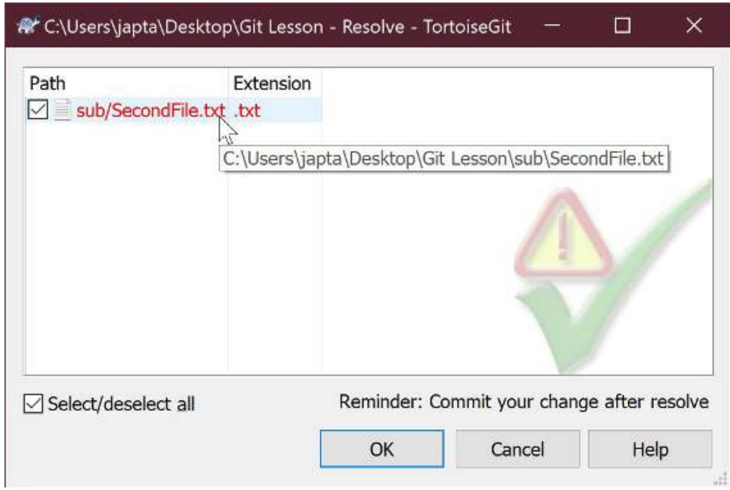
If you want to abort the merge, do a hard reset on HEAD or select abort merge on the context menu.

See help for more information.

Resolving a Merge Conflict

1. Close the instructions on resolving a merge conflict.
2. Another pop-up will appear, asking to review the changes. Click "No" for now.
   a. Note: answering "Yes" to this pop-up brings up a similar dialog.
3. Right-click on an empty part of the file explorer and select, "TortoiseGit -> Resolve..."





Path	Extension
☑ sub/SecondFile.txt	.txt

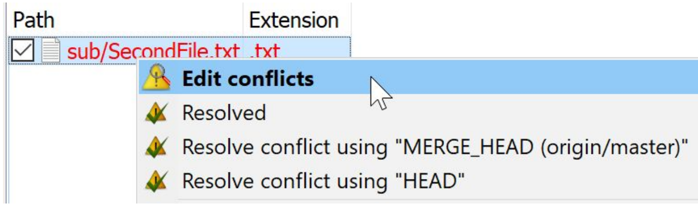
C:\Users\japta\Desktop\Git Lesson\sub\SecondFile.txt

☑ Select/deselect all

Reminder: Commit your change after resolve

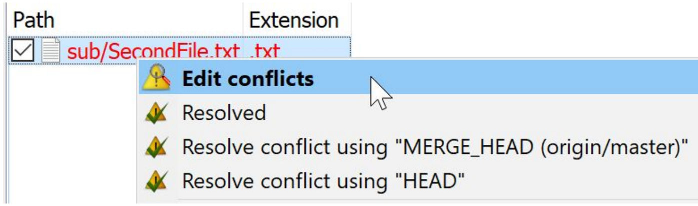
Resolving a Merge Conflict

4. Right-click on "`sub/SecondFile.txt`" line.
    a. "Edit conflicts" opens an editor to merge the conflicting lines.
    b. 'Resolve conflict using "MERGE_HEAD (origin/master)' overwrites the file with what's in the remote repository.
    c. 'Resolve conflict using "HEAD"' overwrites the file with your local repository's copy.
    d. Double-clicking the line will be the same as selecting "Edit conflicts."
5. Select "Edit conflicts."


Path
Extension
sub/SecondFile.txt
.txt
Edit conflicts
Resolved
Resolve conflict using "MERGE_HEAD (origin/master)"
Resolve conflict using "HEAD"

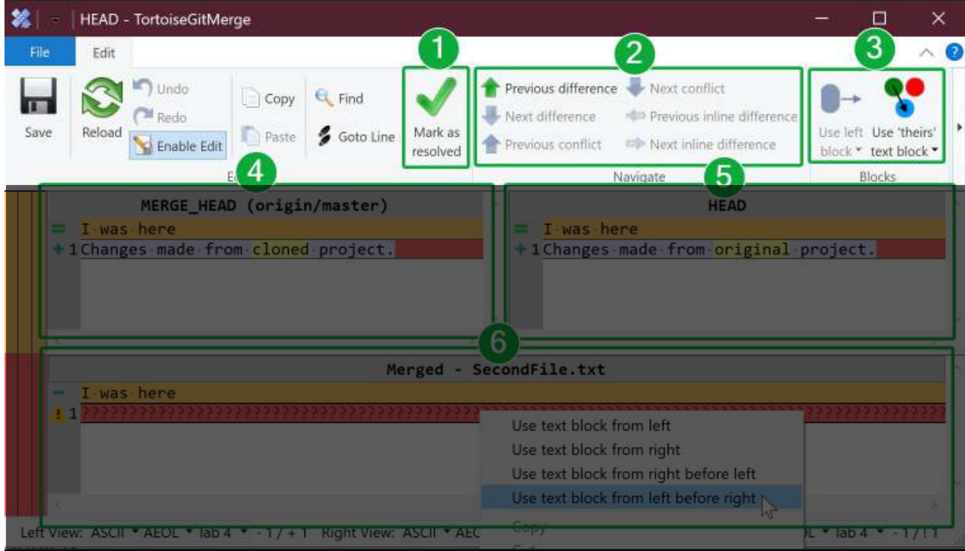
Resolving a Merge Conflict

4. Right-click on "`sub/SecondFile.txt`" line.
    a. "Edit conflicts" opens an editor to merge the conflicting lines.
    b. 'Resolve conflict using "MERGE_HEAD (origin/master)' overwrites the file with what's in the remote repository.
    c. 'Resolve conflict using "HEAD"' overwrites the file with your local repository's copy.
    d. Double-clicking the line will be the same as selecting "Edit conflicts."
5. Select "Edit conflicts."


Path
Extension
sub/SecondFile.txt
.txt
Edit conflicts
Resolved
Resolve conflict using "MERGE_HEAD (origin/master)"
Resolve conflict using "HEAD"

The controls on the top of the Merge dialog has the following functionality:

1. Button to indicate we're done merging the file.
2. Navigation buttons to quickly jump from one conflicting line to another.
3. Multiselect buttons that provides options on how to resolve the highlighted set of conflicting lines. Right-clicking the corresponding lines will provide the same options.



```
MERGE_HEAD (origin/master)
= I was here
+ 1 Changes made from cloned project.
```

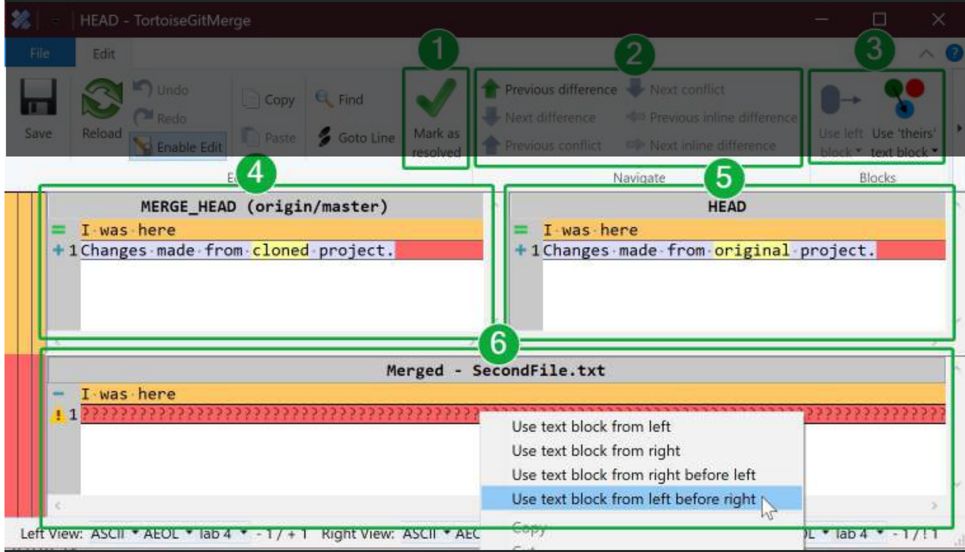
```
HEAD
= I was here
+ 1 Changes made from original project.
```

```
Merged - SecondFile.txt
- I was here
! 1 ?????????????????????????????????????????????????????
```

Left View: ASCII AEOL Tab 4 -1 / +1 Right View: ASCII AEOL Tab 4 -1 / +1

The controls on the bottom of the Merge dialog has the following functionality:

4. Indicates the changes made from the remote repository (MERGE_HEAD (origin/master)).
5. Indicates the changes made from the local repository (HEAD).
6. What the resulting file will look like.



MERGE_HEAD (origin/master)

```
= I was here
+1 Changes made from cloned project.
```

HEAD

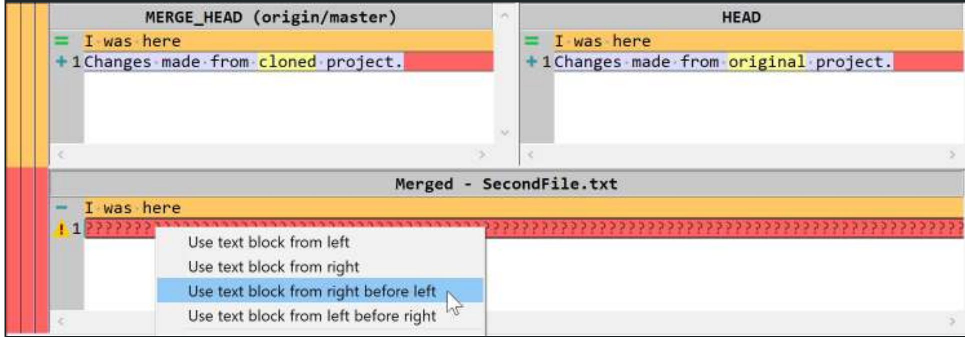
```
= I was here
+1 Changes made from original project.
```

Merged - SecondFile.txt

```
- I was here
! 1 ??????????????????????????????????????????????????????????????????????????
```

Most people use the right-click context menu to resolve conflicts:

1. Right-click the red line in the bottom panel.
2. Select “Use text block from right before left”
   a. Observe how the bottom panel provides a preview of what the merged file would look like.
3. Click, “Mark as resolved”.
4. Close the Merge and Resolve dialog.



Use text block from left
Use text block from right
Use text block from right before left
Use text block from left before right

MERGE_HEAD (origin/master)

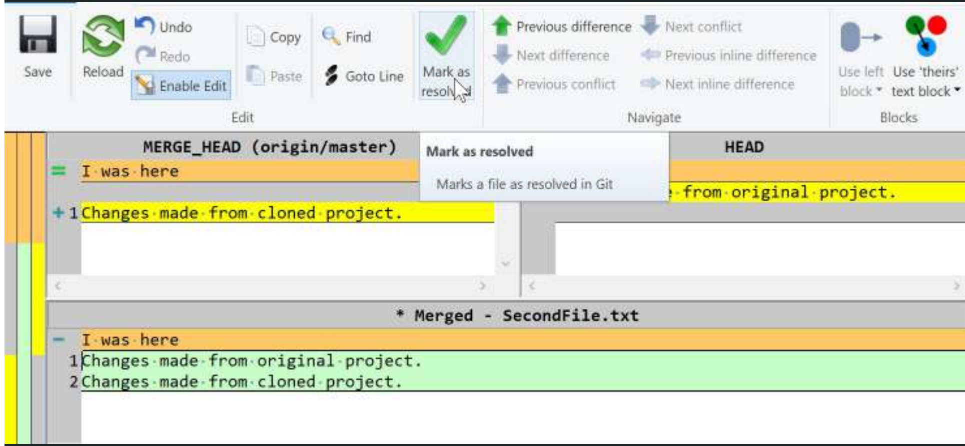
```
= I was here
+1 Changes made from cloned project.
```

HEAD

```
= I was here
+1 Changes made from original project.
```

Merged - SecondFile.txt

```
- I was here
! 1 ??????????????????????????????????????????????????????????????????
```



MERGE_HEAD (origin/master)

```
= I was here
+1 Changes made from cloned project.
```

HEAD

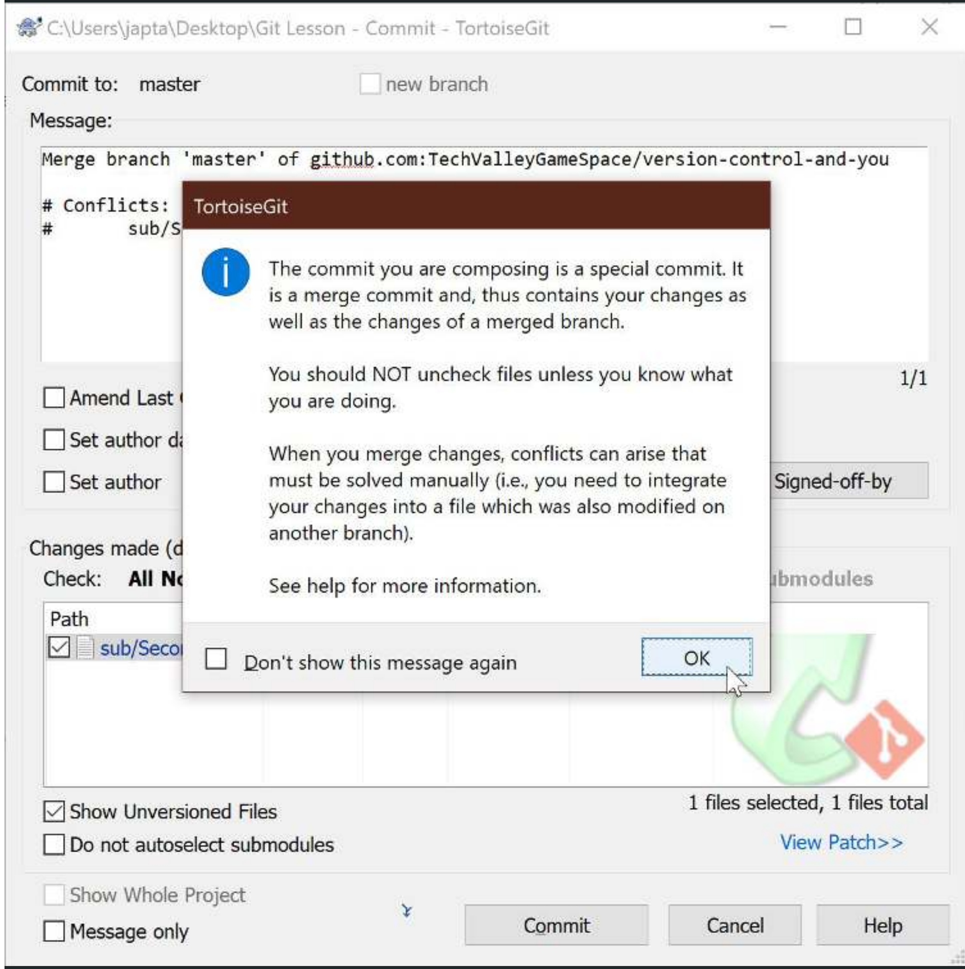
```
from original project.
```

* Merged - SecondFile.txt

```
- I was here
 1 Changes made from original project.
 2 Changes made from cloned project.
```

Don’t forget to commit the changes!

1. On commit, the following dialog may appear.
2. Since we’ve resolved all conflicts, click “OK”.
3. Remove lines that starts with ‘#’ from the message field.
4. Click “Commit”.


C:\Users\japta\Desktop\Git Lesson - Commit - TortoiseGit

Commit to: master ☐ new branch

Message:

```
Merge branch 'master' of github.com:TechValleyGameSpace/version-control-and-you

# Conflicts:
#       sub/S
```

1/1

☐ Amend Last
☐ Set author d
☐ Set author

Signed-off-by

Changes made (d

Check: All N

ubmodules

Path

☑ sub/Seco

☑ Show Unversioned Files
☐ Do not autoselect submodules

1 files selected, 1 files total

View Patch>>

☐ Show Whole Project
☐ Message only

TortoiseGit

The commit you are composing is a special commit. It is a merge commit and, thus contains your changes as well as the changes of a merged branch.

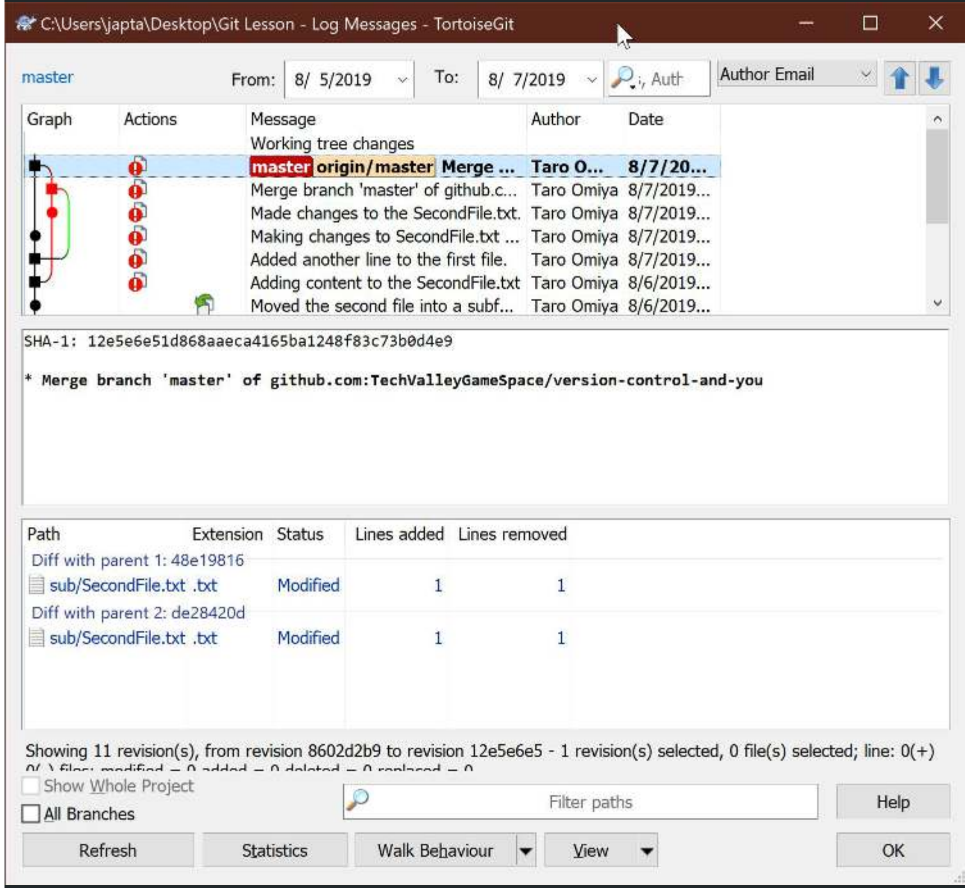
You should NOT uncheck files unless you know what you are doing.

When you merge changes, conflicts can arise that must be solved manually (i.e., you need to integrate your changes into a file which was also modified on another branch).

See help for more information.

☐ Don't show this message again

Notice the log indicates a merge has occurred.



C:\Users\japta\Desktop\Git Lesson - Log Messages - TortoiseGit

master From: 8/ 5/2019 To: 8/ 7/2019 Auth Author Email

Graph	Actions	Message	Author	Date
		Working tree changes		
		master origin/master Merge ...	Taro O...	8/7/20...
		Merge branch 'master' of github.c...	Taro Omiya	8/7/2019...
		Made changes to the SecondFile.txt.	Taro Omiya	8/7/2019...
		Making changes to SecondFile.txt ...	Taro Omiya	8/7/2019...
		Added another line to the first file.	Taro Omiya	8/7/2019...
		Adding content to the SecondFile.txt	Taro Omiya	8/6/2019...
		Moved the second file into a subf...	Taro Omiya	8/6/2019...

```
SHA-1: 12e5e6e51d868aaeca4165ba1248f83c73b0d4e9

* Merge branch 'master' of github.com:TechValleyGameSpace/version-control-and-you
```

Path	Extension	Status	Lines added	Lines removed
Diff with parent 1: 48e19816				
sub/SecondFile.txt	.txt	Modified	1	1
Diff with parent 2: de28420d				
sub/SecondFile.txt	.txt	Modified	1	1

Showing 11 revision(s), from revision 8602d2b9 to revision 12e5e6e5 - 1 revision(s) selected, 0 file(s) selected; line: 0(+)

Questions?