

## 2. Version control

Data Management  
Spring & Summer 2018  
OSIPP, Osaka U

Shuhei Kitamura

# Goal

Get to know about version control!

# Outline

- What is version control?
  - Why do we need it?
- Install Git
- Install SourceTree
- Make a new repository
- Techniques
  - Commit
  - Clone
  - Pull
  - Push
  - Make branches
  - Checkout
  - Merge
  - Rebase
  - Stash
- Appendix: References

# What is version control?



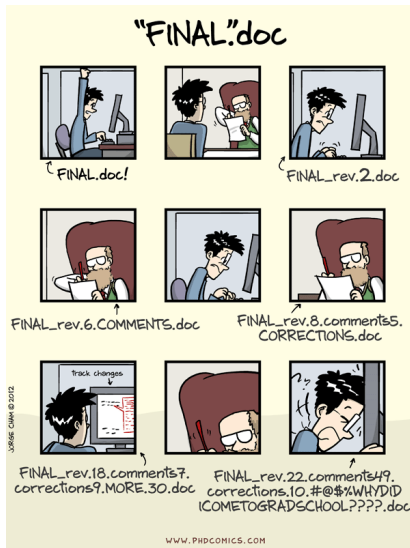
- Starting with a base version, the system keeps track of the changes you made at each step of the way.
- This means that you can always go back to an old version whenever you want.

# What is version control? (cont.)

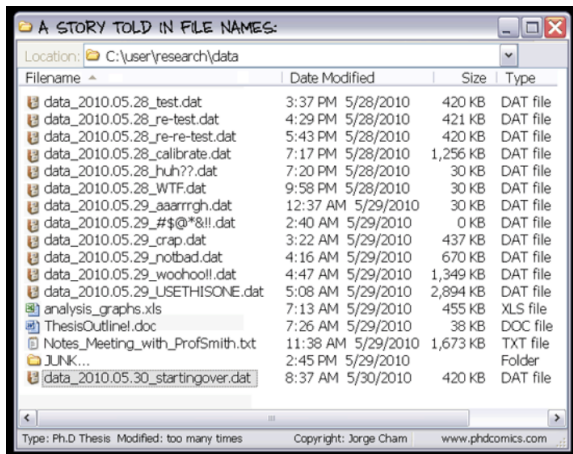


- In team, each member updates the introduction, model, and results section of a co-authored paper separately but simultaneously.
- You can also do it by yourself.
  - You typically make a branch when there is a new issue (e.g. “add references to intro”, “replace all  $\alpha$  in the model as  $\phi$ ”). (We will talk about branching later.)

# Why do we need version control?



## Why do we need version control? (cont.)



# Why do we need version control? (cont.)

You may not be sure which version you need to look at when you restart the project.

- “Okay, there are final.doc, final\_v2.doc, final\_v2\_revised.doc.... Which one should I use?”

With version control, you keep using the same file (and the same file name), but you can always go back to any older version.

- The system keeps all the information of who has updated the file, when and what has been updated!



# Why do we need version control? (cont.)

When you work in team, you may often encounter the following situation:

- Me: "Hey, I've updated the tex file."
- My coauthor: "Oops, sorry I've overwritten yours with mine, which was based on the old version I had. I didn't know that you were also updating the file simultaneously!"
- Me: "..."

Another commonly observed situation:

- Me: "Hey, I find `clean_data_myname.do` and `clean_data_yourname.do`. Which one did we use in the end?"
- My coauthor: "Don't remember."

# Why do we need version control? (cont.)

All these problems will be solved once you and your team members work with version control.

- Version control is useful for both individual work and group work!

Here we will use Git - a very popular version control system for programmers

- Another option: Mercurial

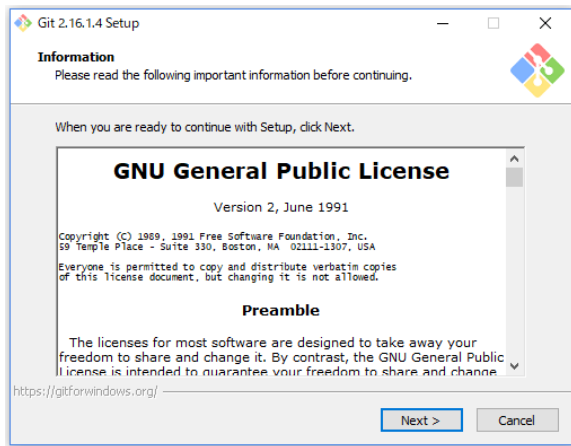
Why do we study version control before studying Python or R?

- I expect you to be familiar with version control by learning-by-doing.

# Set up

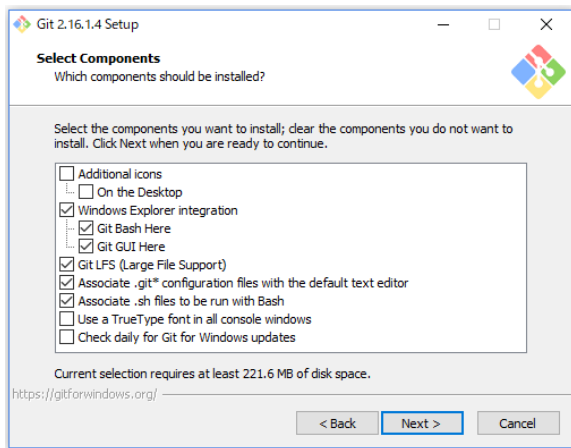
1. Download & install [Git](#)
  - The following instruction is mainly for Windows users.
  - For Mac users, use [Git for Mac](#) or [Homebrew](#).
2. Sign up & create a new repository in [GitHub](#)
  - The largest host of source code in the world.
  - Everything is open. Private repos are not free.
  - Another option: Bitbucket. Private repos are free for up to five team members. Mercurial is also supported.
3. Download & install [SourceTree](#) (Git GUI)
  - Other options: GitKraken, GitHub Desktop

# Install Git



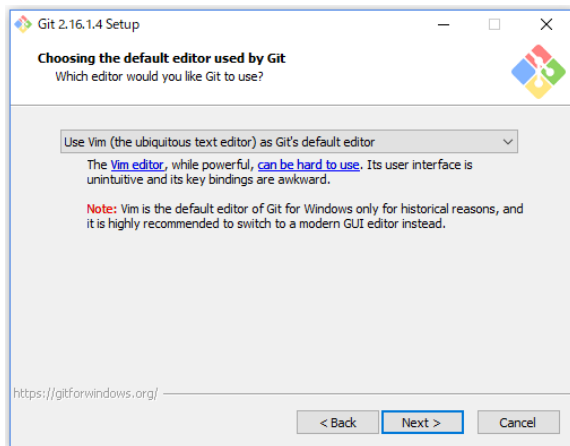
Click next.

# Install Git (cont.)



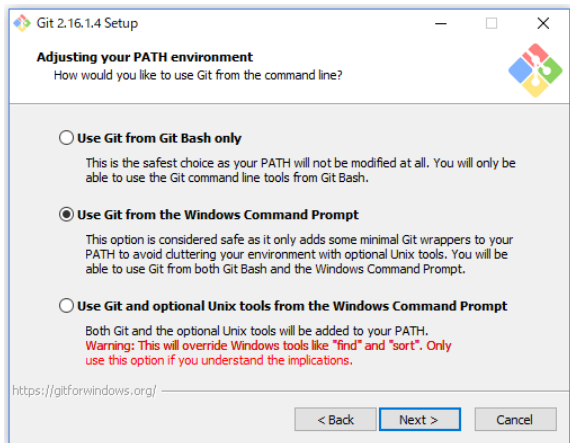
Click next.

# Install Git (cont.)



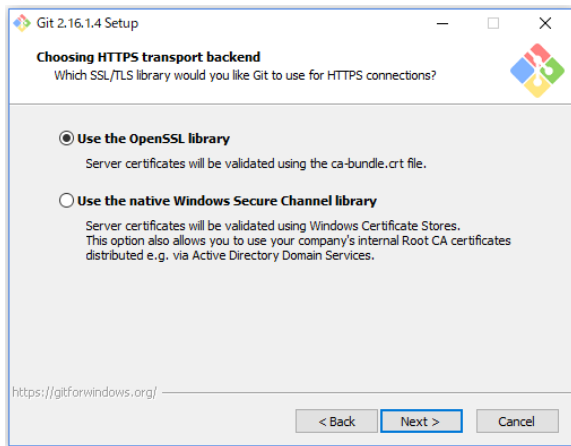
Click next.

# Install Git (cont.)



Click next.

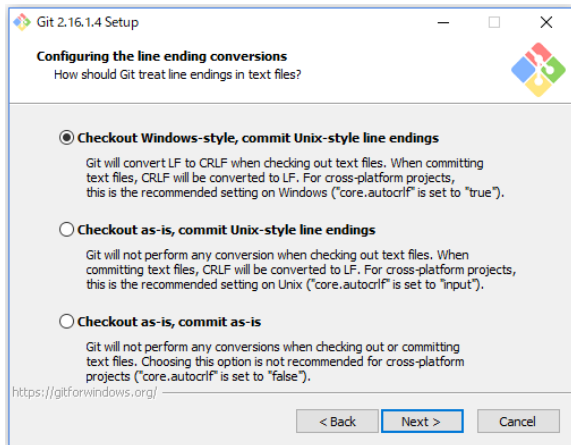
## Install Git (cont.)



Click next.

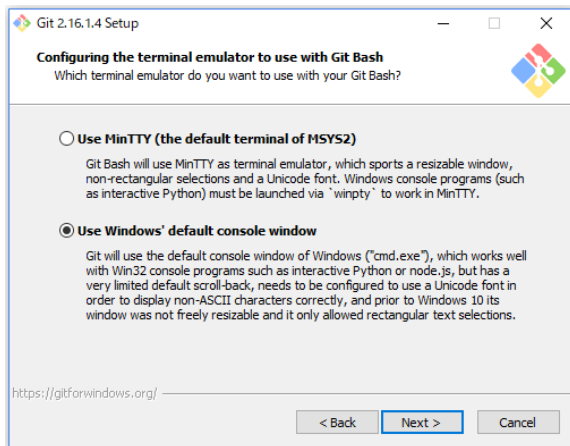


# Install Git (cont.)



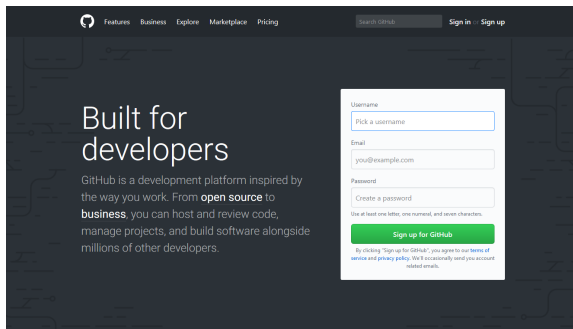
Click next.

# Install Git (cont.)



Click install. → Done!

# Sign up with GitHub



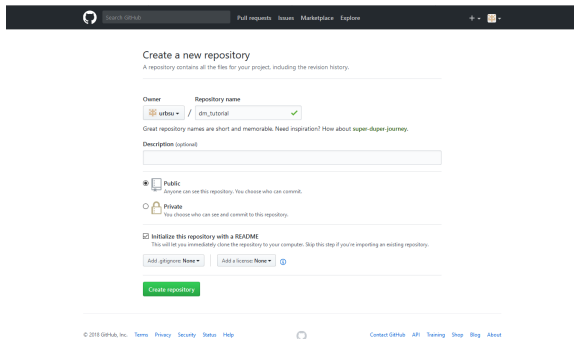
The screenshot shows the GitHub homepage with a dark background and a white sign-up form on the right. The form contains the following fields and elements:

- Username:** A text input field with the placeholder text "Pick a username".
- Email:** A text input field with the placeholder text "you@example.com".
- Password:** A text input field with the placeholder text "Create a password".
- Sign up for GitHub:** A green button.
- Footer text:** "By clicking 'Sign up for GitHub', you agree to our [terms of service](#) and [privacy policy](#). We'll occasionally send you account related emails."

The background of the page features the text "Built for developers" and a description of GitHub as a development platform.

- Once you fill in your username, email address and password, you will get an email to your registered address for verification.
- After clicking Verify email address, you will be asked to make your account name. → Done!
- Next, let's make a repository.

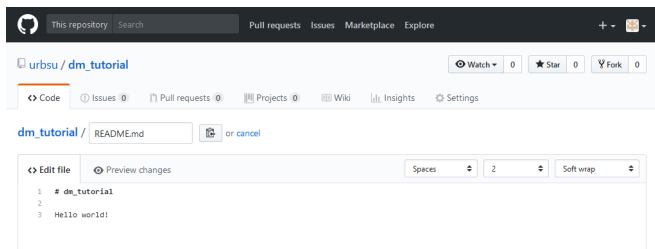
# Make a repository



The screenshot shows the GitHub 'Create a new repository' interface. At the top is a dark navigation bar with the GitHub logo, a search bar, and links for Pull requests, Issues, Marketplace, and Explore. The main heading is 'Create a new repository' with a subtext: 'A repository contains all the files for your project, including the revision history.' Below this, there are two input fields: 'Owner' (set to 'urbou') and 'Repository name' (set to 'dm\_tutorial' with a green checkmark). A 'Description (optional)' text area is below. Two radio buttons are present: 'Public' (selected) and 'Private'. A checkbox 'Initialize this repository with a README' is checked. At the bottom, there are buttons for 'Add a file', 'Add a license', and a green 'Create repository' button. The footer contains copyright information, links to Terms, Privacy, Security, Status, and Help, the GitHub logo, and links to Contact GitHub, API, Training, Shop, Blog, and About.

- Click Start a project.
- Make a repository name (e.g. “dm\_tutorial”)
- Check public.
- Include a README? → Yes.

# Let's edit a README file



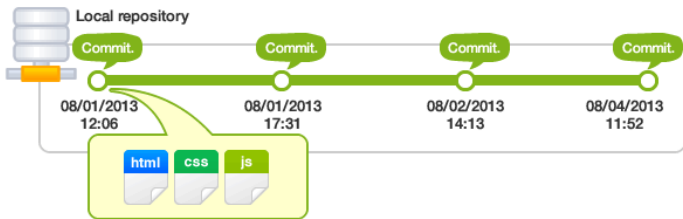
- Click README.md.
- Click a pen icon (edit this file).
- Write “Hello world!” in line 3.
- Write e.g. “Add a line in README.md.” in the comment box at the bottom.
- Click Commit changes.

# After commit

The screenshot shows the GitHub interface for the repository 'urbsu / dm\_tutorial'. At the top, there's a navigation bar with links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below this, the repository name is displayed along with 'Watch', 'Star', and 'Fork' buttons, each with a count of 0. A secondary navigation bar includes 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Insights', and 'Settings'. The main content area shows the 'dm\_tutorial / README.md' file, indicating it was updated by 'urbsu' 303c895 3 minutes ago. Below this, it states '1 contributor'. The file content is displayed in a light gray box, showing the title 'dm\_tutorial' and the text 'Hello world!'. At the bottom of the file view, there are buttons for 'Raw', 'Blame', and 'History', along with icons for a new file, edit, and delete.

- Now README.md is updated.

# What is commit?



You commit when you want to keep the record of your changes.

- It is like taking a picture.
- Commit  $\neq$  save.

A 40-character checksum hash is used to identify a commit.

- Tip: Do not do all the different changes at once in a single commit. Separating commits makes it easy for you and your team members to understand what changes have been made and why.

## When should I commit?

- Commit when you want to “take a picture.”

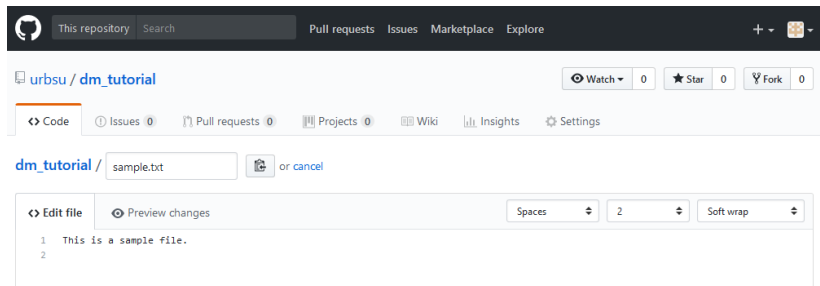


# What should I write in a commit message?

```
1st line: Abstract of the contents changed by commits  
2nd line: Blank line  
3rd line and the following lines: Reason for changes
```

- Write what has been changed, and why.
- Try make it simple and self-telling as much as possible:
  - Abstract: "Replace all  $\alpha$  with  $\phi$  in the model section."
  - Reason: " $\alpha$  has also been used in the empirical section."
- GitHub already has two separated boxes, one for abstract and the other for reason.
- Next, let's make a new file in the repository.

# Make a new file



- Go back to the top level (“dm\_tutorial”)
- Click Create new file. Type a file name (e.g. “sample.txt”)
- Type something in the file (e.g. “This is a sample file.”).
- Write “Create sample.txt” in the comment box at the bottom.
- Click Commit changes.

# After commit

The screenshot shows the GitHub interface for a repository named 'urbsu / dm\_tutorial'. At the top, there's a dark header with the GitHub logo, a search bar, and navigation links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below this, the repository name is displayed with icons for 'Watch' (0), 'Star' (0), and 'Fork' (0). A secondary bar contains links for 'Code', 'Issues' (0), 'Pull requests' (0), 'Projects' (0), 'Wiki', 'Insights', and 'Settings'. The main content area shows 'No description, website, or topics provided.' with an 'Add topics' link and an 'Edit' button. Below this, a summary bar indicates '3 commits', '1 branch', '0 releases', and '1 contributor'. A row of buttons includes 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. The commit history table shows three entries: the initial commit 'urbsu Create sample.txt' (latest, 425fa36, 6 minutes ago), 'Update README.md' (15 minutes ago), and 'Create sample.txt' (6 minutes ago).

Commit Message	Commit Hash	Time Ago
urbsu Create sample.txt	425fa36	6 minutes ago
Update README.md		15 minutes ago
Create sample.txt		6 minutes ago

- Now you have two files in your repository.
- Next, let's clone the repository to your local machine.

# Clone

Clone is like copying a remote repository to your local machine.

- You cannot clone files selectively. You have to clone the entire repository.

With clone, changes you made locally can be reflected in the remote repository.

- So clone  $\neq$  download.


In case you just want to copy a remote repository to your GitHub account, use fork instead.

- You cannot reflect changes you made to the original repository unless you get permission from the owner of the original repository.
- We will talk about fork later.


You can also download the repository as a zip-file.

To clone, let's install SourceTree, a popular Git GUI.

# Install SourceTree

 **ATLASSIAN** 製品 チーム向け サポート

無料評価版 購入する

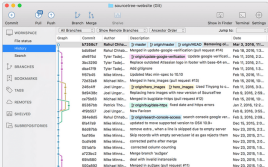
 **SourceTree**

Windows 向けダウンロード

美しい Git GUI で  
シンプルかつパワ  
フルに

Windows 向けダウンロード

Mac OS X 向けも利用可能



## Windows と Mac に対応した Git 無料クライアント

SourceTree により Git リポジトリの操作が簡単になるのでコードを書くことに集中できます。SourceTree のシンプルな GUI を用いてリポジトリを視覚化し管理しましょう。

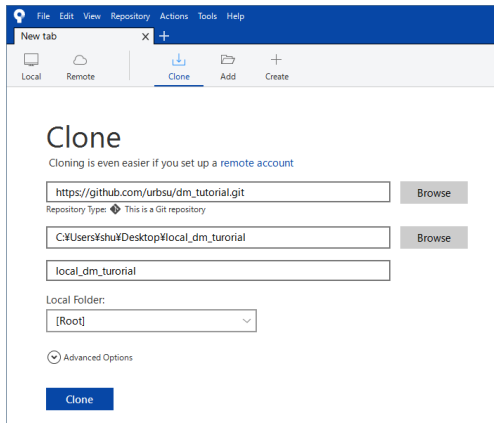
- Install SourceTree on your laptop.
- Make a local folder (e.g. “dm\_tutorial”).

# Clone a repository

The screenshot shows the GitHub interface for the repository 'urbsu / dm\_tutorial'. At the top, there's a navigation bar with 'This repository', 'Search', 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below this, the repository name is displayed with 'Watch', 'Star', and 'Fork' buttons, each with a count of 0. A secondary bar shows 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Insights', and 'Settings'. The main content area has a message 'No description, website, or topics provided.' and an 'Edit' button. Below this, statistics show '3 commits', '1 branch', '0 releases', and '1 contributor'. A bar contains 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and a green 'Clone or download' button. A dropdown menu is open from the 'Clone or download' button, showing 'Clone with HTTPS' (selected), 'Use SSH', and the URL 'https://github.com/urbsu/dm\_tutorial.git'. Below the URL are buttons for 'Open in Desktop' and 'Download ZIP'. A 'Copy to clipboard' tooltip is visible over the 'Download ZIP' button. The repository files list includes 'urbsu Create sample.txt', 'README.md Update README.md', 'sample.txt Create sample.txt', and 'README.md'.

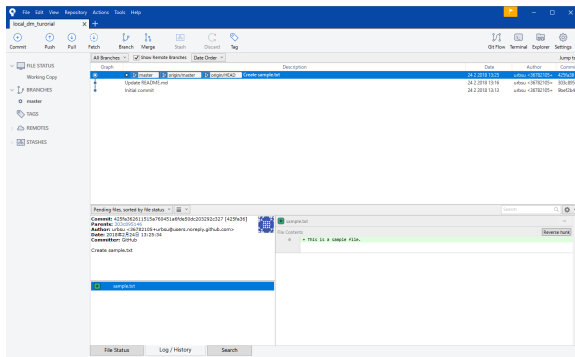
- Go to your repository (“dm\_tutorial”) in GitHub again.
- Click the green button (Clone or download).
- Copy the web URL to clipboard.

# Clone a repository (cont.)



- Start SourceTree. Click File → Clone / New...
- Paste the URL. Select your local folder in the second box.
- Click Clone.

# Clone a repository (cont.)



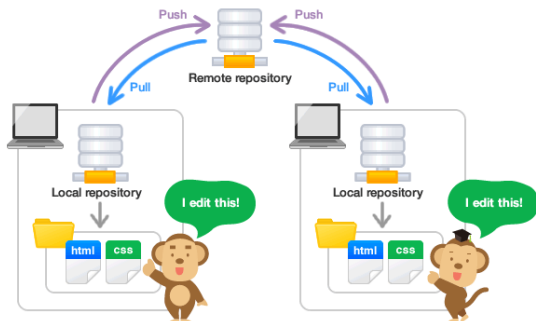
Done! Now you can find two files in the local folder.

Also, you see a connected line (or a graph) in SourceTree.

- Your commits (“Add a line in README.md.” and “Create sample.txt.”) are recorded as two dots in the graph.



# Two types of repositories



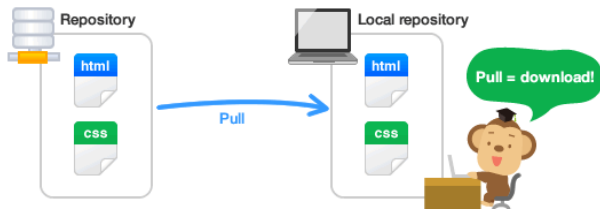
There are two types of Git repositories:

- Remote repository (in GitHub)
- Local repository (on your computer)

Process: (1) cloning a repository, (2) work locally, and (3) reflect changes in the remote repository.

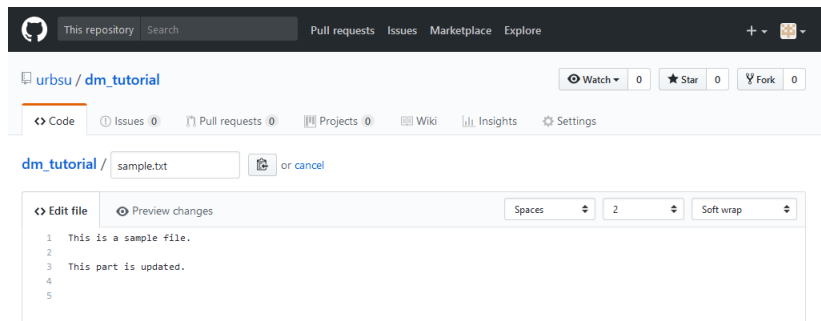
Next, we will learn how to “pull” and “push.”

# Pull



- Pull is like downloading updates from the remote repository.
- Let's make a change in the remote repository.
  - In GitHub, open the "sample.txt" file, then click the pen icon.

# Edit a file in the remote repo



- Write something in line 3 (e.g. “This part is updated.”).
- Write “Add a line in sample.txt.” in the comment box.
- Then commit.

## Edit a file in the remote repo (cont.)

The screenshot shows the GitHub interface for the repository `urbsu / dm_tutorial`. At the top, there's a navigation bar with links for Pull requests, Issues, Marketplace, and Explore. Below this, the repository name is displayed along with icons for Watch (0), Star (0), and Fork (0). A secondary navigation bar includes links for Code, Issues (0), Pull requests (0), Projects (0), Wiki, Insights, and Settings. The main content area shows a message: "No description, website, or topics provided." with an "Edit" button and a link to "Add topics". Below this, a summary bar indicates 4 commits, 1 branch, 0 releases, and 1 contributor. A row of buttons includes "Branch: master", "New pull request", "Create new file", "Upload files", "Find file", and "Clone or download". The commit history table shows two entries: "urbsu Update sample.txt" (latest commit 9816f0a, just now) and "urbsu Update README.md" (an hour ago). The file list below shows `README.md` and `sample.txt`, both with "Update" status and timestamps.

urbsu / dm\_tutorial

Watch 0 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

No description, website, or topics provided. [Edit](#)

[Add topics](#)

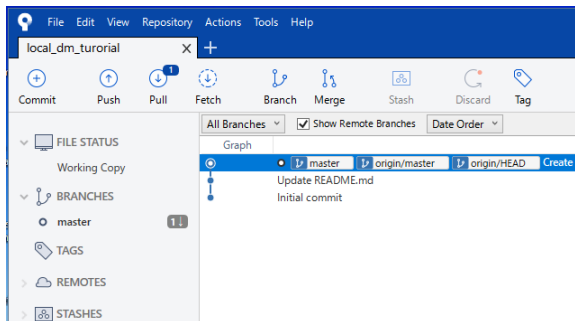
4 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

urbsu Update sample.txt		Latest commit 9816f0a just now
README.md	Update README.md	an hour ago
sample.txt	Update sample.txt	just now

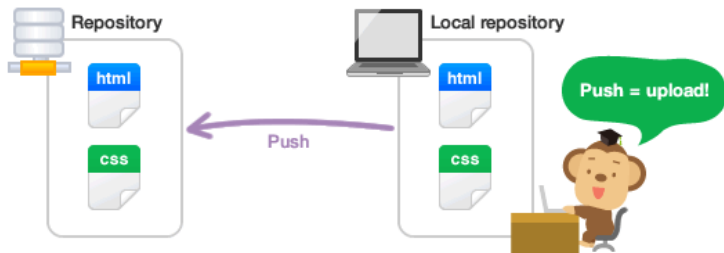
- The commit message has been changed from “Create sample.txt” to “Add a line in sample.txt.”
- Next, let’s pull the latest commit history to your local machine. Go back to SourceTree.

## Pull (cont.)



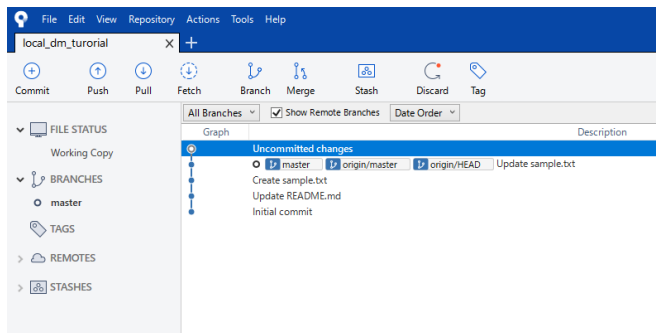
- Now you see Pull on top with “1,” which indicates that there is one additional commit in the remote repository.
- Click Pull. A window pops up. Do not change anything. Click OK.
- Check sample.txt in your local folder. The file should be updated.
- Also, the graph in SourceTree should get additional commit history.
- Next, let's push changes in the local repository to the remote one.

# Push



- Push is like uploading updates from the local repository.
- Let's make a change in the local repository.

# Edit a file in the local repository



- Open the “sample.txt” file in your local folder.
- Add a line in line 5 (e.g. “This part is added in my local repository.”). Then save and close it.
- You should see “Uncommitted changes” in SourceTree.

## Stage and commit first, and push

First of all, the file has to be staged on the index.

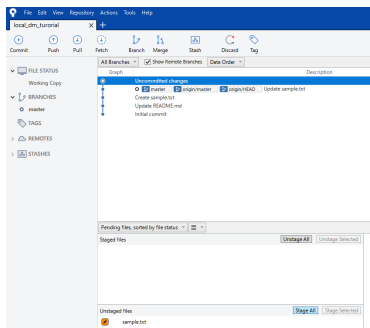
- You can select files you want to commit.

Then you commit changes.

Finally, you push changes to the remote repository.

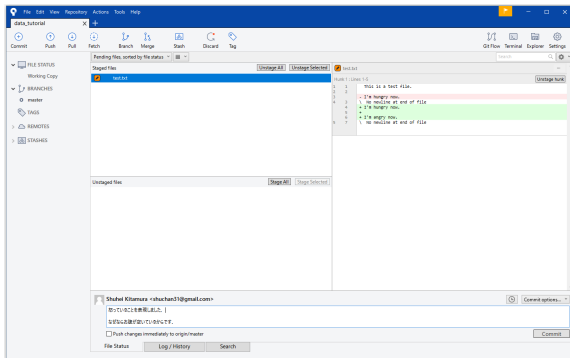


# Stage a file



- Click Stage All at the bottom.
  - In case you want to stage some files but not all of them, select appropriate ones and click Stage Selected.
  - If you stage and commit files one-by-one, you get commit history for each file. This is recommended.
- You can also unstage files if you change your mind.
- Click Commit on top.

# Commit



- Write a commit message “Add a line in sample.txt.”
- Check Push changes immediately to origin/master. This means that local updates will be automatically reflected in the remote repository.
- Click Commit.

# After push

The screenshot shows the GitHub interface for the repository `urbsu / dm_tutorial`. The top navigation bar includes links for Pull requests, Issues, Marketplace, and Explore. The repository name is displayed with icons for Watch (0), Star (0), and Fork (0). Below the repository name, the 'Code' tab is selected, showing the file `dm_tutorial / sample.txt`. A commit message `urbsu Update sample.txt` is visible, along with the commit hash `e5e30f6` and the time `3 minutes ago`. The file content is displayed with line numbers 1 through 5. The file size is 89 Bytes.

urbsu / dm\_tutorial

Watch 0 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Branch: master dm\_tutorial / sample.txt Find file Copy path

urbsu Update sample.txt e5e30f6 3 minutes ago

1 contributor

5 lines (3 sloc) | 89 Bytes Raw Blame History

```
1 This is a sample file.
2
3 This part is updated.
4
5 This part is added in my local repository.
```

- Voila! The file is updated in GitHub.

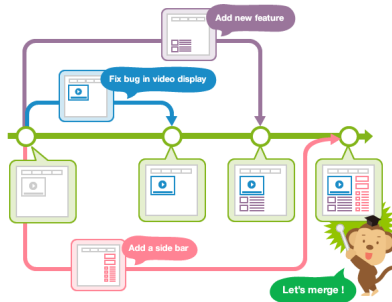
# Summary

What we have learned.

- Commit
- Clone
- Pull
- Push

Next, we will learn how to make branches, checkout, merge/rebase, and stash.

# Branch

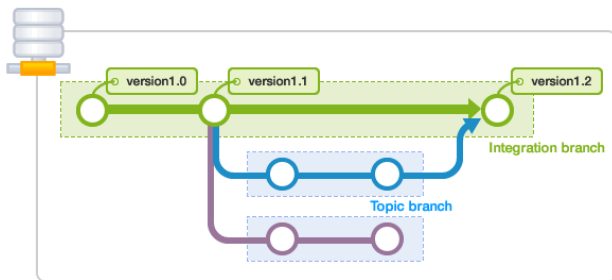


So far, the tree has only one branch (= the master branch). Having several branches makes life easier. E.g. When you work in team, you and your team members often work simultaneously:

- Person A updates the literature review.
- Person A also updates the model section.
- Person B updates the data section.

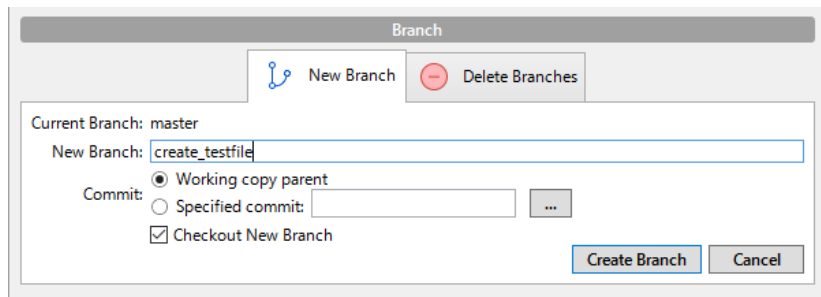
Each task may be more tractable if there are three branches.

## Branch (cont.)



- Integration branch: a stable branch. The master branch is usually used as an integration branch.
- Topic branch: A topic branch represents a task. It is created off from, and merged back into an integration branch. The feature branch plays a role of a topic branch.
- Next, let's make a (feature) branch!

# Make a branch



- Start SourceTree. Click Branch.
- Write a branch name (e.g. "create\_testfile").
- Check Checkout New Branch.
  - You move to a new branch when the branch is created.
- You see "create\_testfile" in BRANCHES to the left. Also, "create\_testfile" appears in the tree.
  - "origin/xxx" means "xxx" in the remote repository.

# What is checkout?



Before checkout

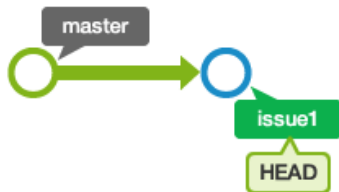


After checkout

- When you create a branch (issue1 in the picture), HEAD is still at master.
- When you checkout, HEAD moves from master to the branch.
- You checkout when you move to the branch that you work on.



## Make a branch (cont.)

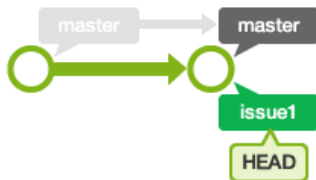


- Suppose that you are at the branch you just created.
- Make a new file (e.g. “test.txt”) in your local repository, and write something in the file (e.g. “This is a test file.”). Save and close.
- In SourceTree, stage the new file, then commit (commit message: e.g. “Create test.txt.”) without push.
  - Make sure you uncheck Push changes immediately to origin/master.
- Double-click “create\_testfile” in BRANCHES. You see that “create\_testfile” is ahead of master.

## Make a branch (cont.)

- Because you have created test.txt and committed changes in the create\_testfile branch, this file should appear only in that branch.
- To see this, open your local repository. Confirm that you have test.txt in that folder.
- Next, double-click master in BRANCHES in SourceTree. Go back to the same local folder again. Can you still find test.txt there?
- In order to reflect your commits to the master branch, you have to merge them into the master branch.
  - You do not need to merge every time you commit. You can merge when you are ready to merge (e.g. when you finish the task).

# Merge



- To merge your commits, you have to be at the master branch. Double-click master in BRANCHES.
- Click Merge on top. When a new window pops up, choose the create\_testfile branch. Then press OK.
- You see that master is now moved up to the create\_testfile branch.
  - This type of merge is called a fast-forward merge.
- Double-click master and open your local repository. You should see test.txt that did not exist before in the folder.

## Merge (cont.)

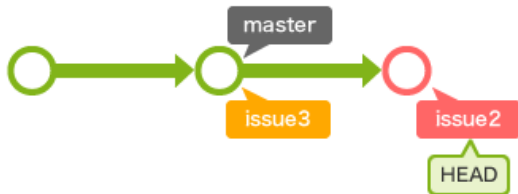
- In SourceTree, you see that “origin/master” and “origin/HEAD” are still one step behind the top of the tree. This is because you have not pushed the changes caused by merge.
- Press Push on top. When a new window pops up, select master and press push.
- Now you see “origin/master” and “origin/HEAD” are also moved to the top of the tree.
- Because you have already merged changes to the master branch, you can delete the feature branch.
- Go to the “master” branch (because you cannot delete a branch when you are at the branch). Right click “create\_testfile.” Select delete create\_testfile.

# Make several branches



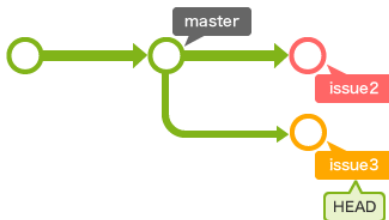
- Let's make two branches, and work in each branch separately.
  - This can happen when a single person has two tasks or two persons have the single task each.
  - For the latter, another person may make a branch in her/his own local machine, and not in yours.
- Make two branches (e.g. “add\_litrev” and “add\_model”) in SourceTree.

## Make several branches (cont.)



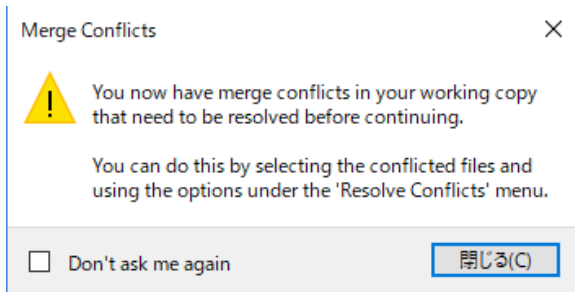
- Double-click “add\_litrev” in BRANCHES.
- In your local repository, open test.txt, and add a line (e.g. “In this section, I will write a literature review.”). Save and close.
- Next, stage and commit (commit message = e.g. “Add a line in the lit review.”) without push in SourceTree.

## Make several branches (cont.)



- Double-click “add\_model” in BRANCHES and open test.txt. The change you made in the “add\_litrev” branch does not yet appear here.
- Add a line (e.g. “In this section, I will write an economic model.”). Save and close.
- Stage and commit (commit message = e.g. “Add a line in the model section.”) without push in SourceTree.
- Voila! Now you see two branches!
- Next, let’s merge these two branches into the master branch.

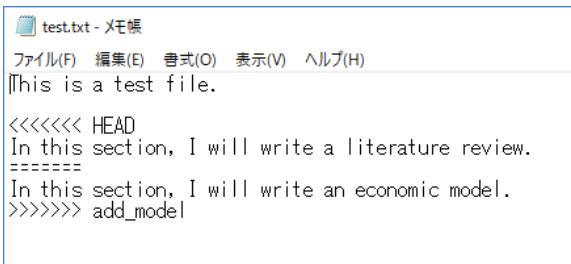
# Merge



- Double-click master in BRANCHES and click Merge. Choose “add\_litrev”, and merge.
- Now master has moved to the position of “add\_litrev”, which means that the branch has been merged with master. A fast-forward merge.
- Next, double-click master again and click Merge. This time choose “add\_model”, and merge.
- A new window pops up telling you that you have merge conflicts.



## Merge (cont.)

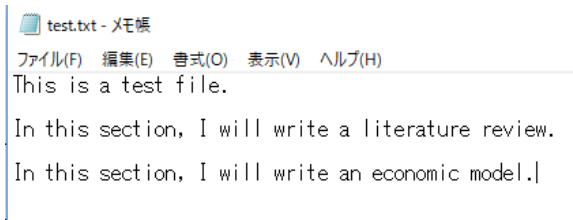


```
test.txt - メモ帳
ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)
This is a test file.

<<<<<<< HEAD
In this section, I will write a literature review.
=====
In this section, I will write an economic model.
>>>>>>> add_model
```

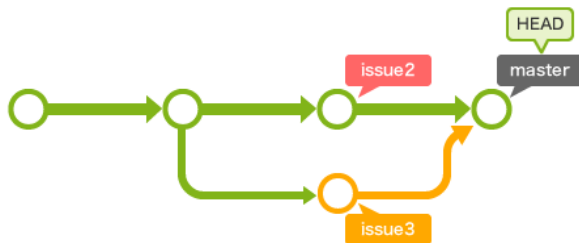
- When you get such a message, you need to resolve the conflicts by yourself.
- Go to the local repository, and open test.txt. The file may look like the above picture.
- The line after “<<<<<<< HEAD” is included in the master branch (which has been merged with “add\_litrev”), but not in the “add\_model” branch.
- The line after “=====” is included in the “add\_model” branch, but not in the master branch.

## Merge (cont.)



- Modify the file as in the picture. Save and close.
- Stage and commit. You should be able to merge this time.

## Merge (cont.)

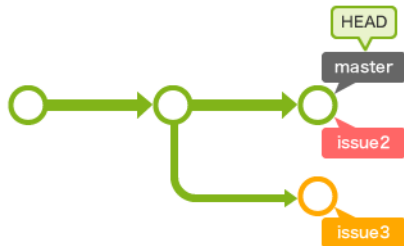


- Now the revision history should look like this.
  - In particular, master should be at the top of the tree.
- A new commit for merging “add\_model” (issue3 in the picture) has been added.
  - This is not a fast-forward merge, called a non fast-forward merge.

## Merge (cont.)

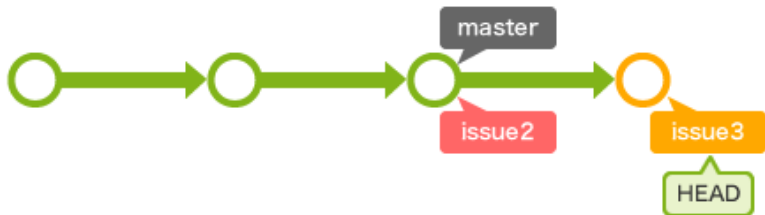
- So far, we have considered merge conflicts when you push.
- The same thing can happen when you pull.
  - The procedure is very similar. You have to resolve the conflict, and then commit.
- Instead of pull, you may use fetch for checking updates in the remote repository.
  - Pull = fetch + merge.

# Rebase



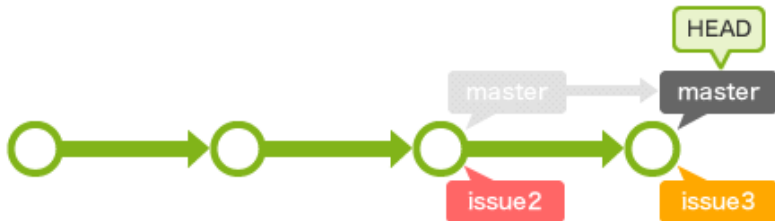
- Merge is a method to merge commit histories.
- Another method is rebase.
- Right click “Add a line in the lit review” in SourceTree. Click Reset current branch to this commit.
- Choose “Hard” for Using mode.
- When you open test.txt. You should only see “In this section, I will write a literature review.” We have returned back to the “add\_litrev” before merge.

## Rebase (cont.)



- Checkout “add\_model”.
- Right click on master. Choose Rebase current changes onto master.
- A window pops up. Click OK. You will get an error message of merge conflict.
- Resolve merge conflict by opening test.txt.
- In Action tab on top, click Continue Rebase.
- Commit history should look like the above figure (“add\_model” (issue3) is ahead of master).

## Rebase (cont.)



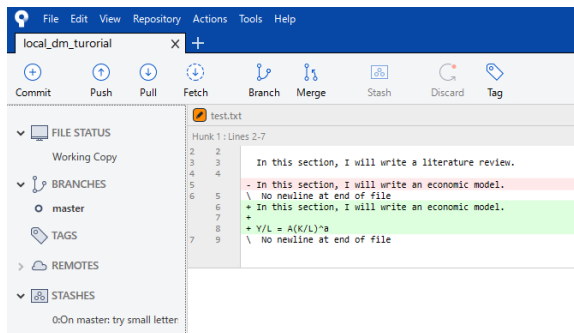
- Checkout master.
- Merge "add\_model".
- Finally, delete two feature branches, and push changes to the remote repository.

# Stash

- Suppose you have started a new task after creating a feature branch (e.g. update the model section).
- Suddenly, you feel that you need to go back to the original version to do something else (e.g. write a model with different notations).
- However, you feel that you do not want to delete the current version or commit.
- In this case, you can use stash. With stash, your current version is saved temporarily.
- Let's try it.



# Stash (cont.)



- Open test.txt the local repository. Write something (e.g. “ $Y/L = A(K/L)^a$ ”). Save and close.
- You realize that you want to try rewriting the model with small letters, while keeping the current one.
- Click Stash on top. A window pops up. Put a name for this stash (e.g. “capital\_letter\_ver”).
- The stash appears in STASHES to the left.

## Stash (cont.)

- Open test.txt. The file has been reverted to the original version (the version of when you committed last time).
- Write something (e.g. "y = Ak^a"). Save and close.
  1. If you keep using the small letter version, just keep writing.
    - You can delete the capital letter version. Right-click the stash in STASHES, and click Delete Stash capital\_letter\_ver.
  2. If you want to switch back to the capital letter version.
    - Right-click the capital\_letter\_stash in STASHES, and click Apply Stash capital\_letter\_ver.
    - This returns an error message indicating merge conflicts.
    - Resolve the conflicts, then commit.
    - Alternatively, stash the small letter version first. Name the stash (e.g. "small\_letter\_ver"). Then apply the stash of the capital letter version. In this case, both versions will remain unless you delete them.
- You may also use stash when you pull updates from the remote repository.

# Summary

What we have learned earlier:

- Commit
- Clone
- Pull
- Push

What we have additionally learned:

- Make branches
- Checkout
- Merge
- Rebase
- Stash

These are the basic features of Git.

## Appendix: References

# References

## Git

- Git Beginner's Guide for Dummies ([English](#)) ([Japanese](#))
- Manga De Wakaru Git ([Japanese](#))