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
    - Fork
- Appendix: References
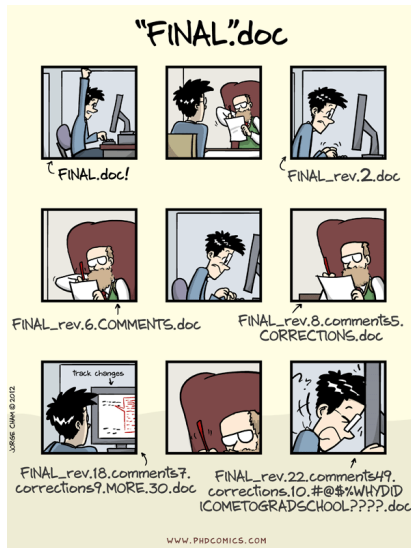
What is version control?

- 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.
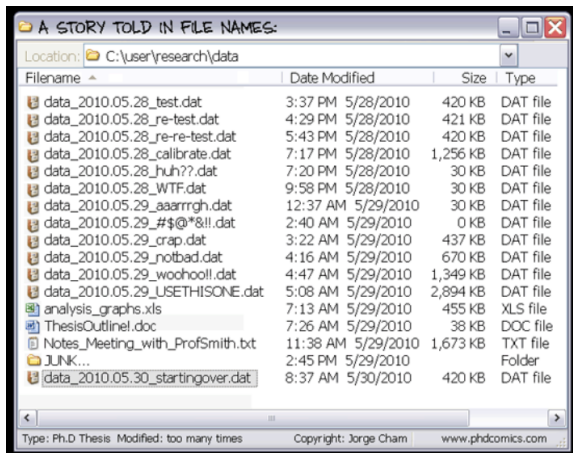
What is version control? (cont.)

- 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 α in the model as ϕ"). (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
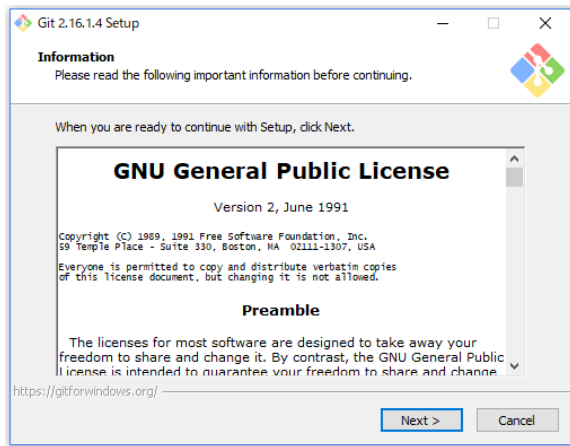
Click next.

Install Git (cont.)
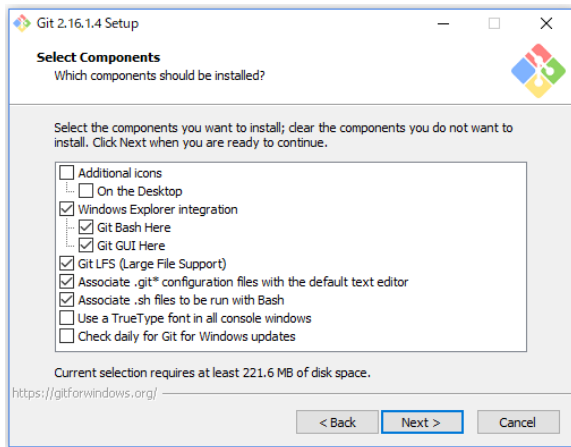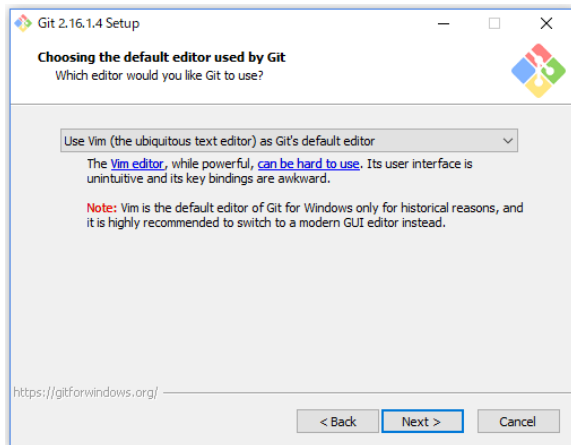
Click next.

Install Git (cont.)

Click next.
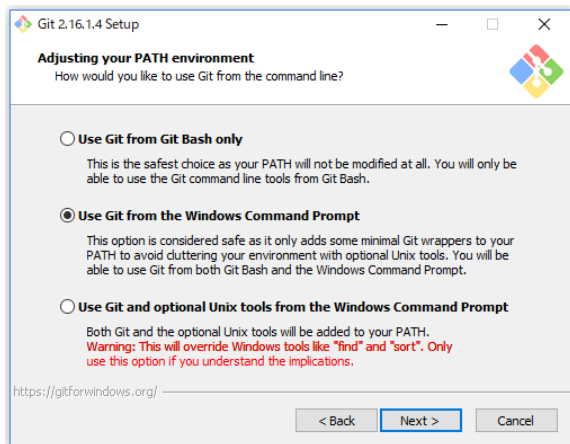
Install Git (cont.)

Click next.
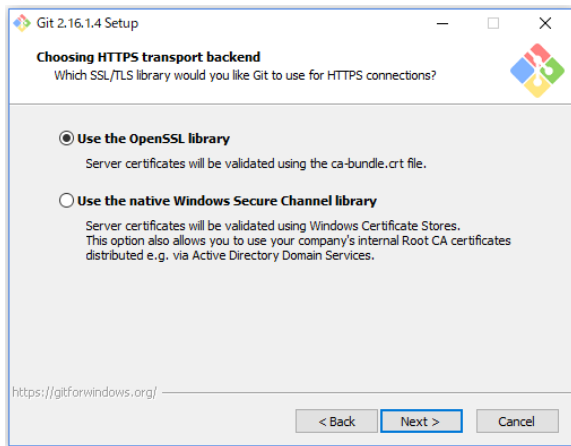
Install Git (cont.)

Click next.
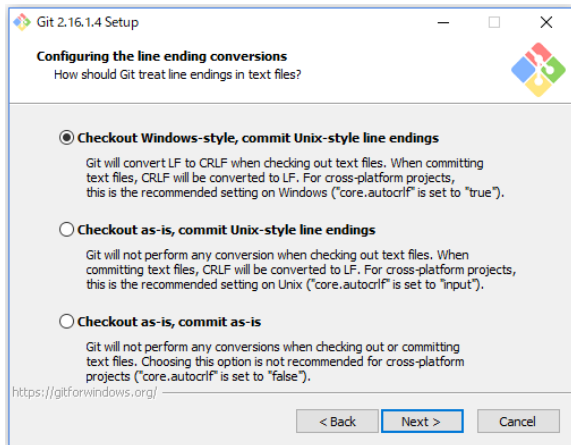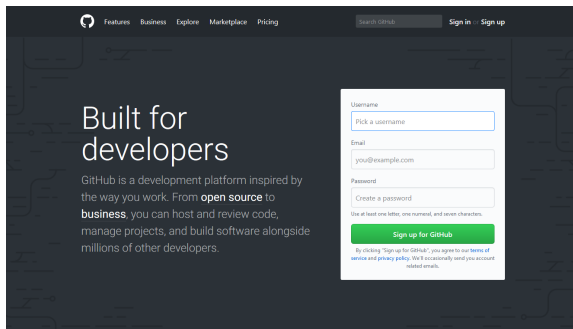
Install Git (cont.)

Click next.

Install Git (cont.)
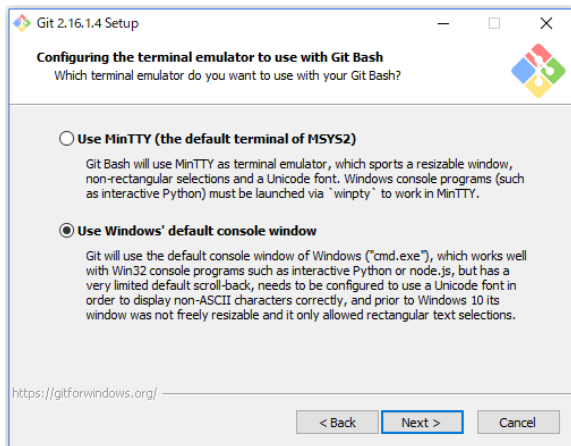
Click install. → Done!

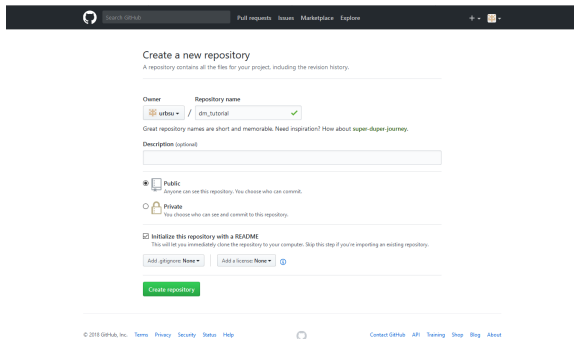# Sign up with GitHub

- 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

- Click Start a project.
- Make a repository name (e.g. "dm_tutorial")
- Check public.
- Include a README? → Yes.

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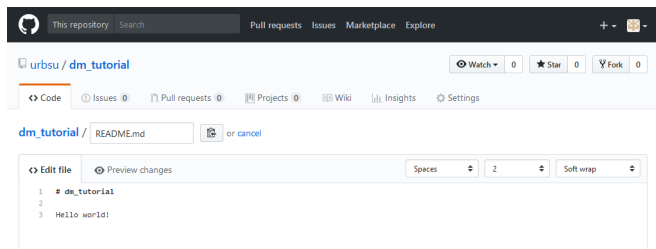
- Click README.md.
- Click a pen icon (edit this file).
- Write "Hello world!" in line 3.
- Write e.g. "Add a line in README.md." in the comment box at the bottom.
- Click Commit changes.

After commit

- Now README.md is updated.

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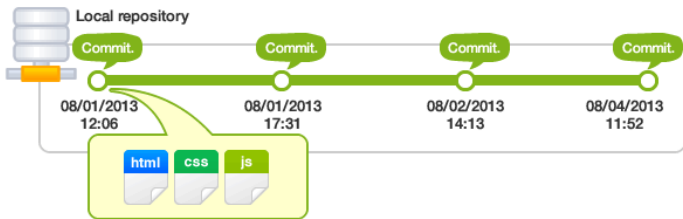
You commit when you want to keep the record of your changes.

- 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 easily 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 α with ϕ in the model section."
    - Reason: "α 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
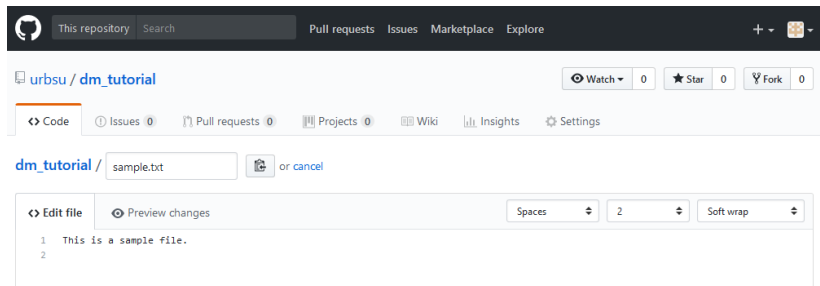
- Go back to the top level ("dm_tutorial")
- Click Create new file. Type a file name (e.g. "sample.txt")
- Type something in the file (e.g. "This is a sample file.").
- Write "Create sample.txt" in the comment box at the bottom.
- Click Commit changes.

After commit

- 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

- Install SourceTree on your laptop.
- Make a local folder (e.g. "dm_tutorial").

Clone a repository

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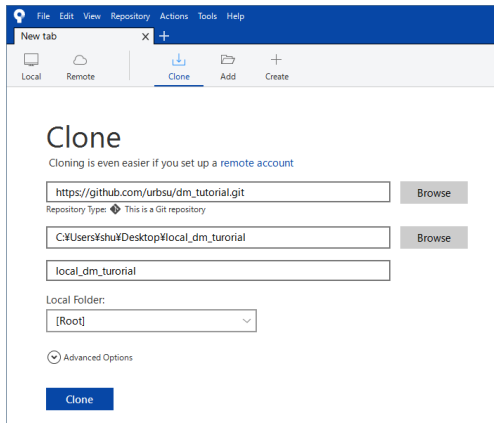
- Start SourceTree. Click File → Clone / New...
- Paste the URL. Select your local folder in the second box.
- Click Clone.

Clone a repository (cont.)
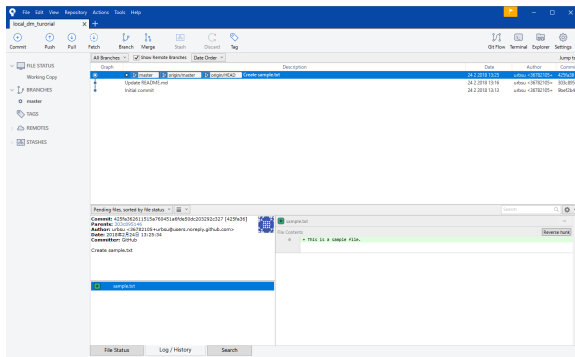
Done! Now you can find two files in the local folder.

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
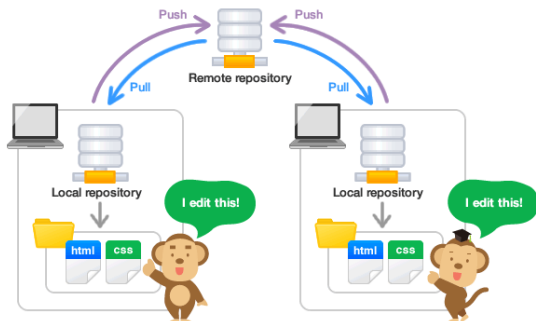
There are two types of Git repositories:
- Remote repository (in GitHub)
- Local repository (on your computer)

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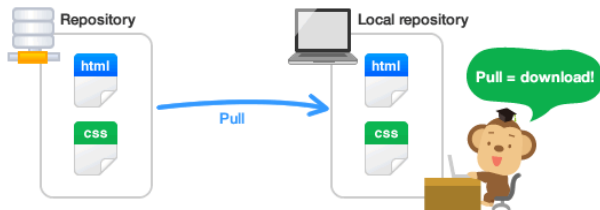
- Pull is like downloading updates from the remote repository.
- Let's make a change in the remote repository.
    - In GitHub, open the "sample.txt" file, then click the pen icon.

Edit a file in the remote repo
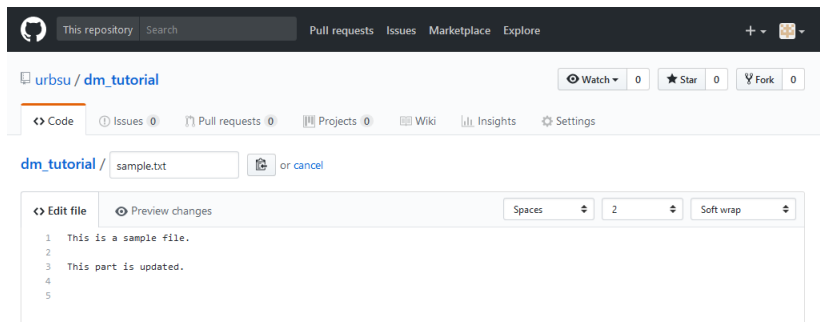
- Write something in line 3 (e.g. "This part is updated.").
- Write "Add a line in sample.txt." in the comment box.
- Then commit.

Edit a file in the remote repo (cont.)

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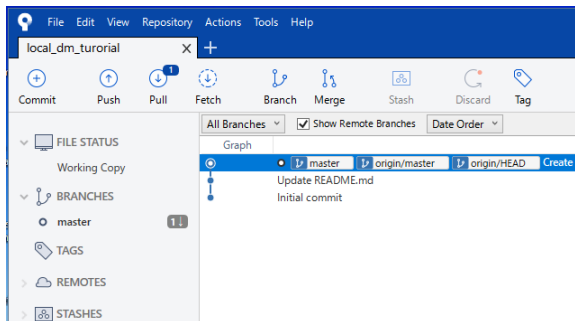
- Now you see Pull on top with "1," which indicates that there is one additional commit in the remote repository.
- Click Pull. A window pops up. Do not change anything. Click OK.
- Check sample.txt in your local folder. The file should be updated.
- Also, the graph in SourceTree should get additional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

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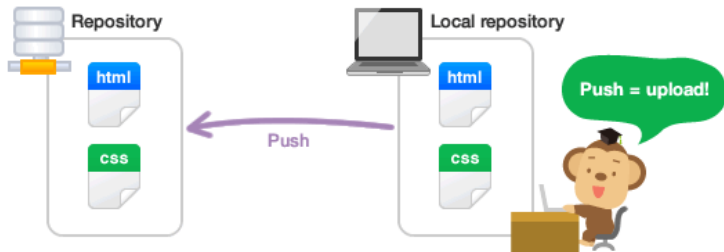
- Push is like uploading updates from the local repository.
- Let's make a change in the local repository.

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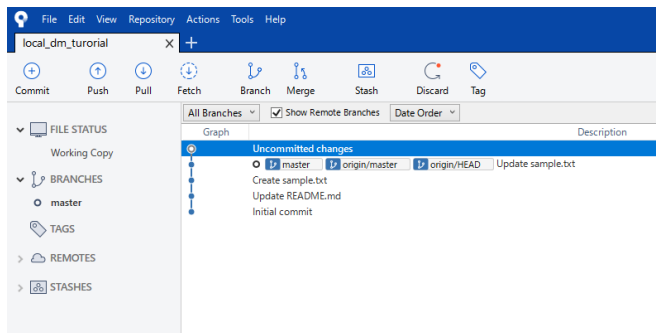
- Open the "sample.txt" file in your local folder.
- Add a line in line 5 (e.g. "This part is added in my local repository."). Then save and close it.
- You should see "Uncommitted changes" in SourceTree.

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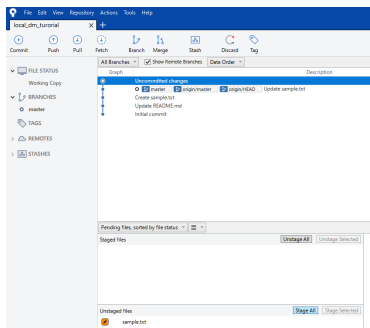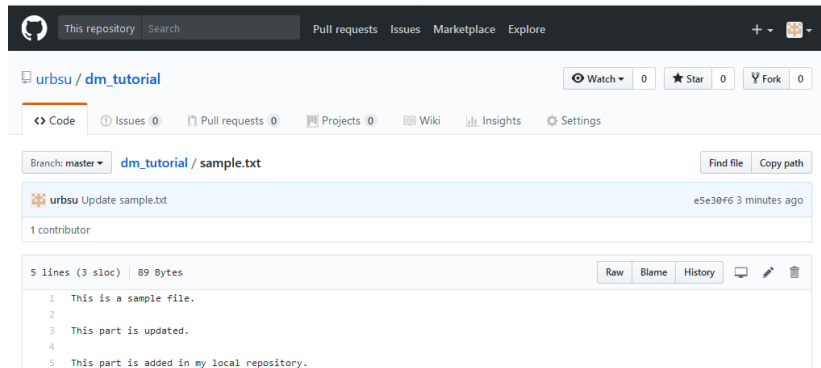
- Click Stage All at the bottom.
    - In case you want to stage some files but not all of them, select appropriate ones and click Stage Selected.
    - If you stage and commit files one-by-one, you get commit history for each file. This is recommended.
- You can also unstage files if you change your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

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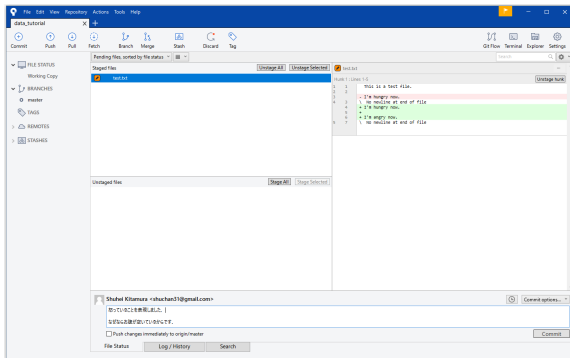
- Write a commit message "Add a line in sample.txt."
- Check Push changes immediately to origin/master. This means that local updates will be automatically reflected in the remote repository.
- Click Commit.

After push

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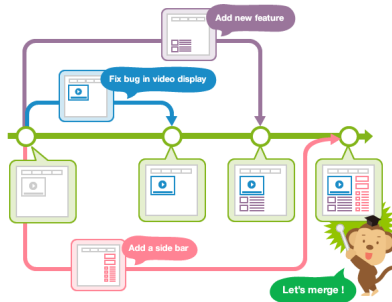
So far, the tree has only one branch (= the master branch).
Having several branches makes life easier. E.g. When you work in team, you and your team members often work simultaneously:

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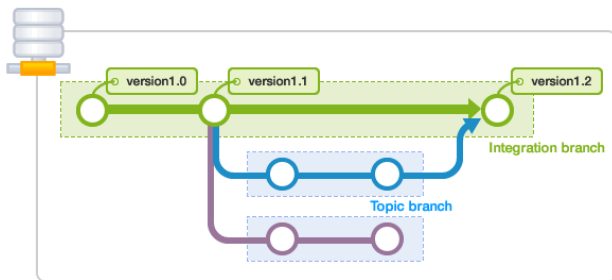
- Integration branch: a stable branch. The master branch is usually used as an integration branch.
- Topic branch: A topic branch represents a task. It is created off from, and merged back into an integration branch. The feature branch plays a role of a topic branch.
- Next, let's make a (feature) branch!

Make a branch
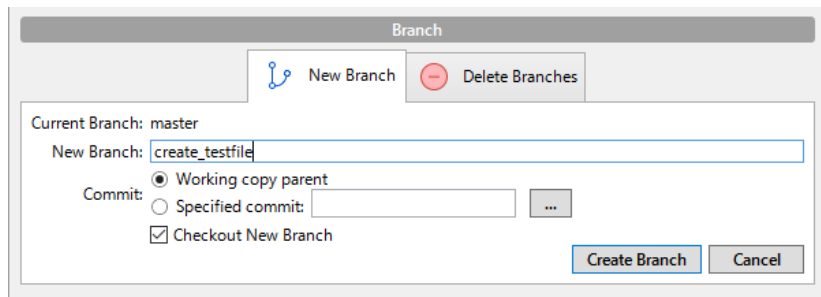
- Start SourceTree. Click Branch.
- Write a branch name (e.g. "create_testfile").
- Check Checkout New Branch.
    - → You move to a new branch when the branch is created.
- You see "create_testfile" in BRANCHES to the left. Also, "create_testfile" appears in the tree.
    - "origin/xxx" means "xxx" 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.)
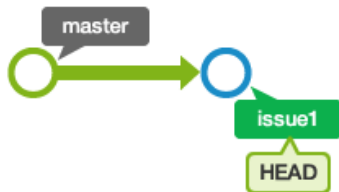
- Suppose that you are at the branch you just created.
- Make a new file (e.g. "test.txt") in your local repository, and write something in the file (e.g. "This is a test file."). Save and close.
- In SourceTree, stage the new file, then commit (commit message: e.g. "Create test.txt.") without push.
    - 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
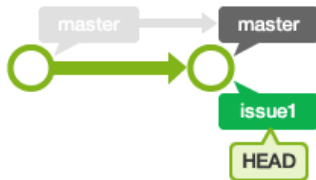
- To merge your commits, you have to be at the master branch. Double-click master in BRANCHES.
- Click Merge on top. When a new window pops up, choose the create_testfile branch. Then press OK.
- You see that master is now moved up to the create_testfile branch.
  - This type of merge is called a fast-forward 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.)
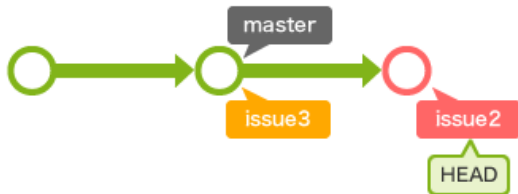
- Double-click "add_litrev" in BRANCHES.
- In your local repository, open test.txt, and add a line (e.g. "In this section, I will write a literature review."). Save and close.
- Next, stage and commit (commit message = e.g. "Add a line in the lit review.") without push in SourceTree.

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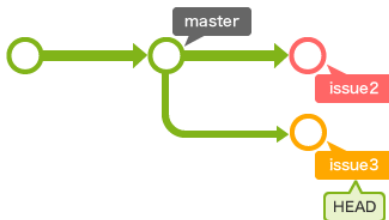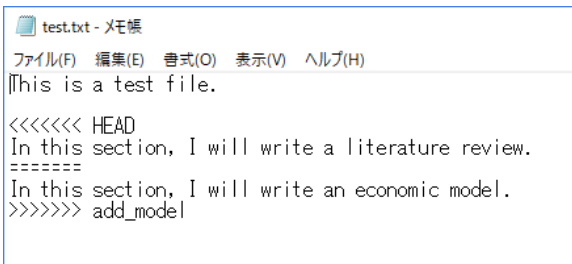
- Double-click "add_model" in BRANCHES and open test.txt. The change you made in the "add_litrev" branch does not yet appear here.
- Add a line (e.g. "In this section, I will write an economic model."). Save and close.
- Stage and commit (commit message = e.g. "Add a line in the model section.") without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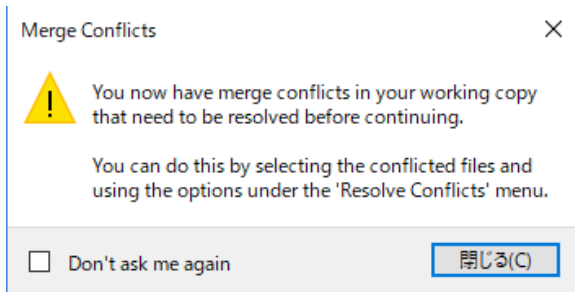
Merge

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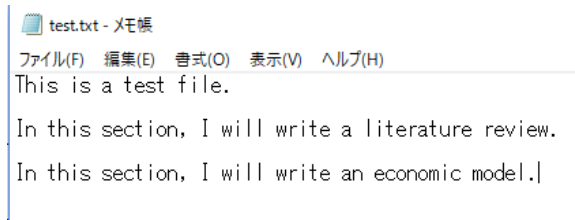
- Modify the file as in the picture. Save and close.
- Stage and commit. You should be able to merge this time.

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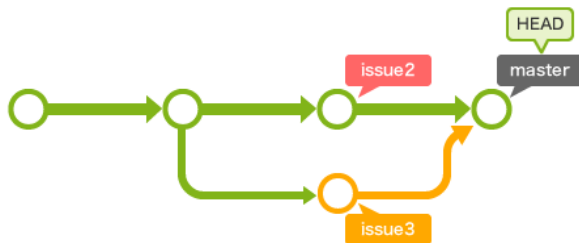
- Now the revision history should look like this.
  - In particular, master should be at the top of the tree.
- A new commit for merging "add_model" (issue3 in the picture) has been added.
  - This is not a fast-forward merge, called a non fast-forward merge.

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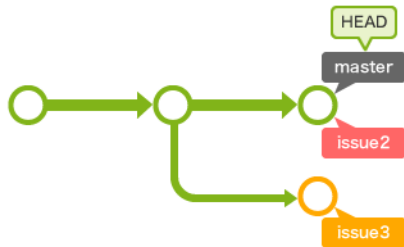
- Merge is a method to merge commit histories.
- Another method is rebase.
- Right click "Add a line in the lit review" in SourceTree. Click Reset current branch to this commit.
- Choose "Hard" for Using mode.
- When you open test.txt. You should only see "In this section, I will write a literature review."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

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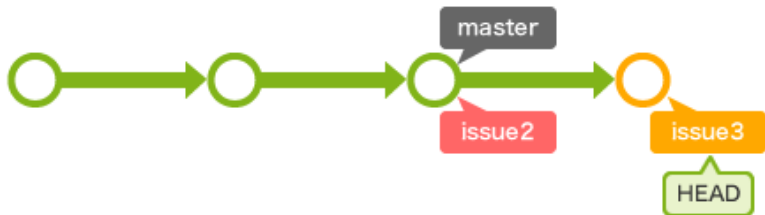
- Checkout "add_model".
- Right click on master. Choose Rebase current changes onto master.
- A window pops up. Click OK. You will get an error message of merge conflict.
- Resolve merge conflict by opening test.txt.
- In Action tab on top, click Continue Rebase.
- Commit history should look like the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

Rebase (cont.)
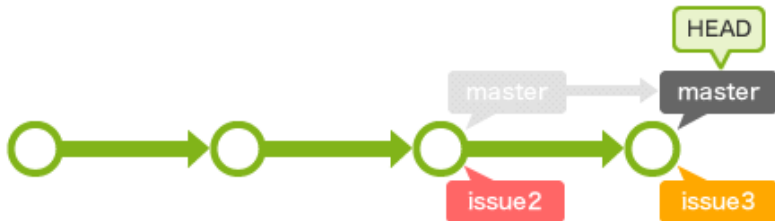
- Checkout master.
- Merge "add_model".
- Finally, delete two feature branches, and push changes to the remote repository.

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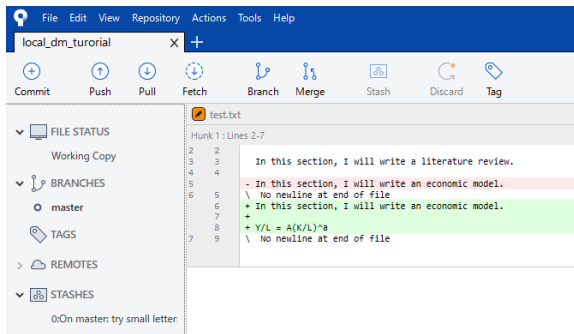
- Open test.txt the local repository. Write something (e.g. "Y/L = A(K/L)^a"). Save and close.
- You realize that you want to try rewriting the model with small letters, while keeping the current one.
- Click Stash on top. A window pops up. Put a name for this stash (e.g. "capital_letter_ver").
- The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

Fork

- Finally, let's fork lecture materials from my GitHub repository using SourceTree.
- Fork creates a copy of a repository under your GitHub ID, while Clone clones the repository to your local machine.
    - If you fork, your update will *not* be reflected back to the original repository unless you get a permission.
- Go to my repository.
- Click "Fork." My repository will be copied to your GitHub page.
    - GitHub might ask you to verify your email address in this process.
- Open SourceTree. Clone the forked repository to your local machine.

Sync a fork

- If there is any update in my repository (e.g., additional lecture slides are uploaded), what happens to your forked repository?
  - It won't be reflected to your forked repository unless you do it manually. So let's do it manually now.
- Open SourceTree. Go to the forked repository.
- In the "Repository" tab on top, choose "Repository Settings."
- In the "Remotes" tab, click "Add."
- Write any remote name (e.g. "upstream"). Paste the path to my GitHub repository (https://github.com/iheus/data_management). Click OK, then OK.
- Click Pull. For Pull from remote, choose upstream, then OK.
  - In case you want to know what changes have been made before merging them, use fetch, instead of pull. Then use merge to merge them.
  - Pull = Fetch + Merge
- Push the changes to your remote repository.

Move a file

- You should have two local repositories:

  (a) Your repository ("dm_tutorial") - Your working repository
  (b) Forked repository ("data_management") - Contain lecture materials

- Let's move an ipynb file from (b) to (a).

  - ipynb files can be read in Jupyter Notebook.
  - We will use the file in the next topic.

- Copy and paste "python_basics.ipynb" from (b) to (a).

  - You can actually continue working on (b) without moving files to (a). However, keep using (a) as your working directory would avoid any confusion.

Summary

What we have learned earlier:

- Commit
- Clone
- Pull
- Push

What we have additionally learned:

- Make branches
- Checkout
- Merge
- Rebase
- Stash
- Fork

These are the basic features of Git.

Appendix: References

References

Git

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