Introduction to open-Source Software (OSS)

Concepts, strategies, and methodologies related to open-source software development

Week 06 – Lecture 09

Jamil Hussain
jamil@sejong.ac.kr
010-6252-8807

Office: 421, Innovation Center
Sejong University

Recap

- Git
  - Git workflow
  - Installation
  - Using Git
  - Configurations
  - Creating Snapshots
  - Tracking

Recap

- Configurations:
  - Customize your environment with git config
  - Three levels: System, Global, Local
- Identity:
  - Set username and email for commits
- **Editor:**
  - Set your preferred editor for Git
- Default Branch:
  - Set the default branch to main

Recap

- Common Commands:
  - Check configurations: git config –list
  - Initialize a repository: git init
  - Check status: git status
  - Stage changes: git add
  - Ignore files: .gitignore

Today, Agenda

- Git
  - Configurations
  - Creating Snapshots
  - Tracking
  - Commit History
  - GitHub Introduction
  - Working with Remotes
  - Branches

세종대학교
SEJONG UNIVERSITY

Viewing Your Staged and Unstaged Changes

- If the **git status** command is too vague for you — you want to know exactly what you changed, not just which files were changed — you can use the **git diff** command.
- It compares what is in your working directory with what is in your staging area.
- If you want to see what you've staged that will go into your next commit, you can use **git diff --staged**. This command compares your staged changes to your last commit:

```
$ git diff
```

```
$ git diff --staged
```

```
$ git diff --cached
```

세종대학교
SEJONG UNIVERSITY

Committing Your Changes

- The simplest way to commit is to type git commit:
- Skipping the Staging Area
    - Adding the -a option to the git commit command makes Git automatically stage every file that is already tracked before doing the commit, letting you skip the git add part

```
$ git commit
```

```
$ git commit -m "Story 182: fix benchmarks for speed"
```

```
$ git commit -a -m 'Add new benchmarks'
```

Skipping the staging area

- If you want to skip the staging area, Git provides a simple shortcut.
- Adding the -a option to the git commit command makes Git automatically stage every file that is already tracked before doing the commit

```
$ git commit -a -m 'Message'
```

```
$ git commit -am ‘Message’
```

Removing Files

- To remove a file from Git, you have to remove it from your tracked files
- The git rm command does that, and also removes the file from your working directory
- To untrack the file **$git rm --cached**

```
$ git rm file1.html
```

```
$ git rm -- cached file1.html
```

세종대학교 SEJONG UNIVERSITY

Viewing the Commit History - 1/3

- After you have created several commits, or
- if you have cloned a repository with an existing commit history, you'll probably want to look back to see what has happened. The most basic and powerful tool to do this is the **git log** command.

```
$ git clone https://github.com/schacon/simplegit-progit
```

```
$ git log
```

```
$ git show #code        # Shows the given commit
```

```
$ git show HEAD         # Shows the last commit
```

Viewing the Commit History -2/3

- A huge number and variety of options to the git log command are available to show you exactly what you’re looking for.
  - One of the more helpful options is -p or --patch, which shows the difference (the patch output) introduced in each commit.

```
$ git log -p -2
```

  - For example, if you want to see some abbreviated stats for each commit, you can use the --stat option:

```
$ git log --stat
```

  - Another really useful option is --pretty.

```
$ git log --pretty=oneline
```

  - The most interesting option value is format, which allows you to specify your own log output format.

```
$ git log --pretty=format:"%h - %an, %ar : %s"
```

Viewing the Commit History -3/3

Specifier Description of Output

%H	Commit hash
%h	Abbreviated commit hash
%T	Tree hash
%t	Abbreviated tree hash
%P	Parent hashes
%p	Abbreviated parent hashes
%an	Author name
%ae	Author email
%ad	Author date (format respects the --date=option)
%ar	Author date, relative
%cn	Committer name
%ce	Committer email
%cd	Committer date
%cr	Committer date, relative
%s	Subject

Option	Description
-p	Show the patch introduced with each commit.
--stat	Show statistics for files modified in each commit.
--shortstat	Display only the changed/insertions/deletions line from the --stat command.
--name-only	Show the list of files modified after the commit information.
--name-status	Show the list of files affected with added/modified/deleted information as well.
--abbrev-commit	Show only the first few characters of the SHA-1 checksum instead of all 40.
--relative-date	Display the date in a relative format (for example, “2 weeks ago”) instead of using the full date format.
--graph	Display an ASCII graph of the branch and merge history beside the log output.
--pretty	Show commits in an alternate format. Option values include oneline, short, full, fuller, and format (where you specify your own format).
--oneline	Shorthand for --pretty=oneline --abbrev-commit used together.

Limiting Log Output

- git log takes a number of useful limiting options;

- The time-limiting options such as --since and --until are very useful. For example, this
- command gets the list of commits made in the last two weeks:

Option	Description
-<n>	Show only the last n commits.
--since, --after	Limit the commits to those made after the specified date.
--until, --before	Limit the commits to those made before the specified date.
--author	Only show commits in which the author entry matches the specified string.
--committer	Only show commits in which the committer entry matches the specified string.
--grep	Only show commits with a commit message containing the string.
-S	Only show commits adding or removing code matching the string.

```
$ git log --pretty="%h - %s" --author='Junio C Hamano' --since="2008-10-01" \ --before="2008-11-01" --no-merges -- t/
```

Undoing Changes in Git

- Git provides various commands to undo changes at different stages of your workflow. These commands help in:
  - **Unstaging files**
  - **Modifying previous commits**
  - **Restoring files**
- **Why It's Important**
  - Mistakes happen! It's crucial to know how to undo actions in Git to avoid problems and preserve your work effectively.

Undoing Changes : git commit --amend

- Purpose
  - Modify the most recent commit.
- Usage Scenario
  - You just made a commit but forgot to include a file or made a mistake in your commit message.
- How It Works
  - Allows you to amend the last commit by adding more changes or editing the commit message.

```
git commit --amend
```

Example→ You realize you forgot to include a file in your last commit. Stage the file, and then run, This will update the last commit, including the new file.

```
git add forgotten-file.js
git commit --amend
```

세종대학교 SEJONG UNIVERSITY

Undoing Changes : git reset HEAD <file>

- Purpose
  - Unstage changes that were accidentally added.
- Usage Scenario
  - You've added files to the staging area but decide that you don't want them included in the commit.
- How It Works
  - Moves files from the staging area back to the working directory, without affecting the file’s contents.

```
git reset HEAD <file>
```

Example→ You added file.js to the staging area, but now you want to unstage it,

```
git reset HEAD file.js
```

The file remains in your working directory but is no longer staged for commit.

Undoing Changes : git restore --staged <file>

- Purpose
  - Another way to unstage files, similar to git reset, but specific to the git restore command introduced in newer versions of Git.
- Usage Scenario
  - Useful in unstaging files when using modern Git workflows with git restore.
- How It Works
  - Removes a file from the staging area.

```
git restore --staged <file>
```

Example→ You decide not to include file.js in the next commit:

```
git restore --staged file.js
```

This removes the file from staging.

Undoing Changes : git checkout -- <file>

- Purpose
  - Discard changes in your working directory and revert a file to its last committed state.
- Usage Scenario
  - You've modified a file but realize you want to revert it to its original version from the last commit.
- How It Works
  - Overwrites the changes in your working directory with the version in the last commit.

```
git checkout -- <file>
```

Example→ You want to discard the changes you made to file.js and restore it to the previous commit:

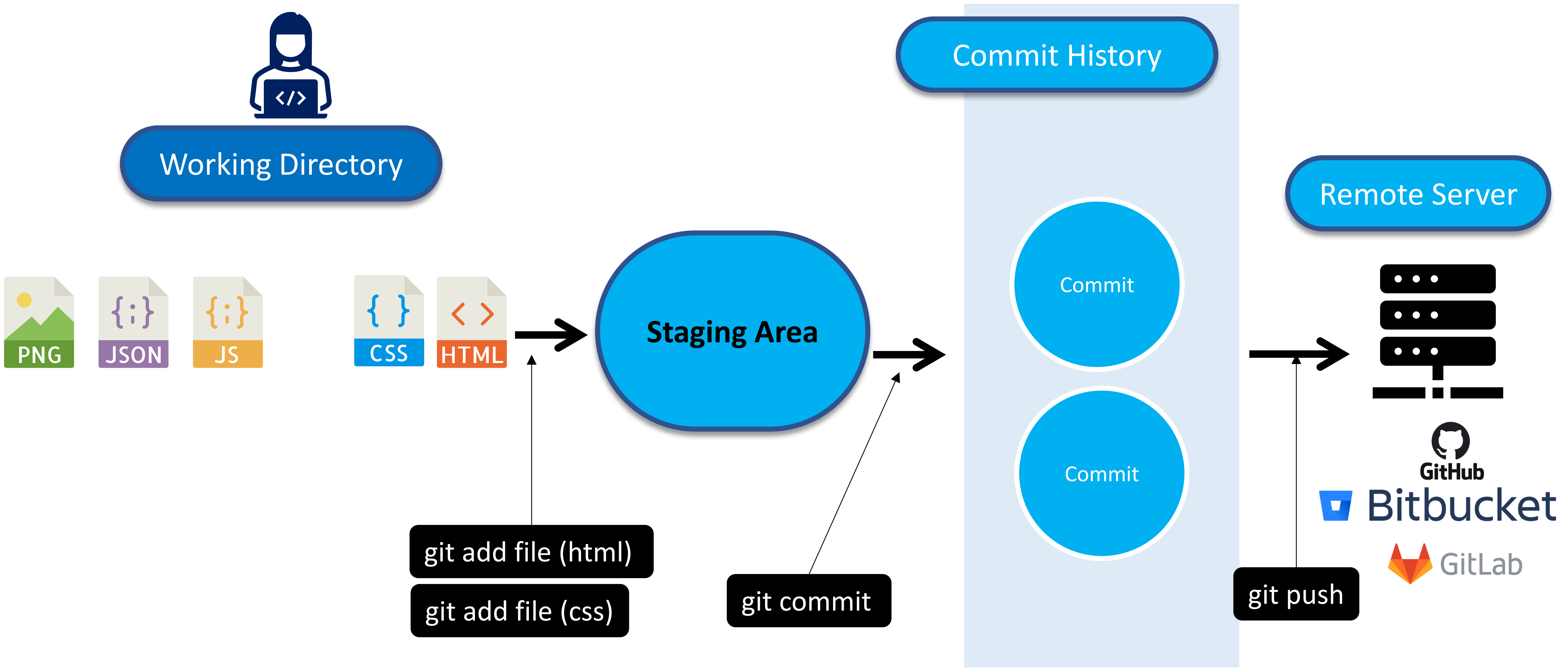
```
git checkout -- file.js
```

This reverts file.js to its last committed state.

Undoing Changes : Summary

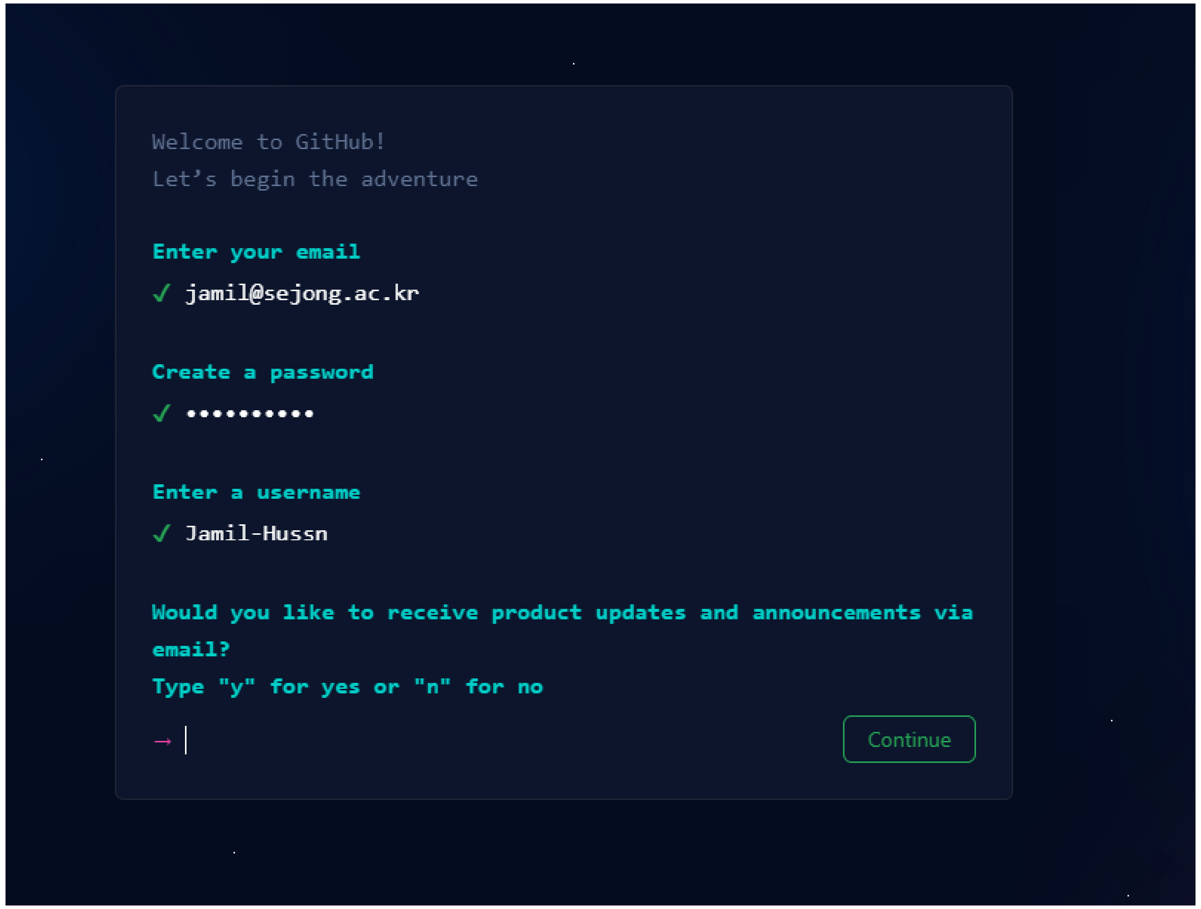
- **git commit --amend:** Edit the last commit by adding new changes or modifying the message.
- **git reset HEAD <file>:** Unstage a file without changing its content.
- **git restore --staged <file>:** Similar to git reset, removes files from the staging area
- **.git checkout -- <file>:** Discards changes in the working directory and restores the file from the last commit.

How Git Works



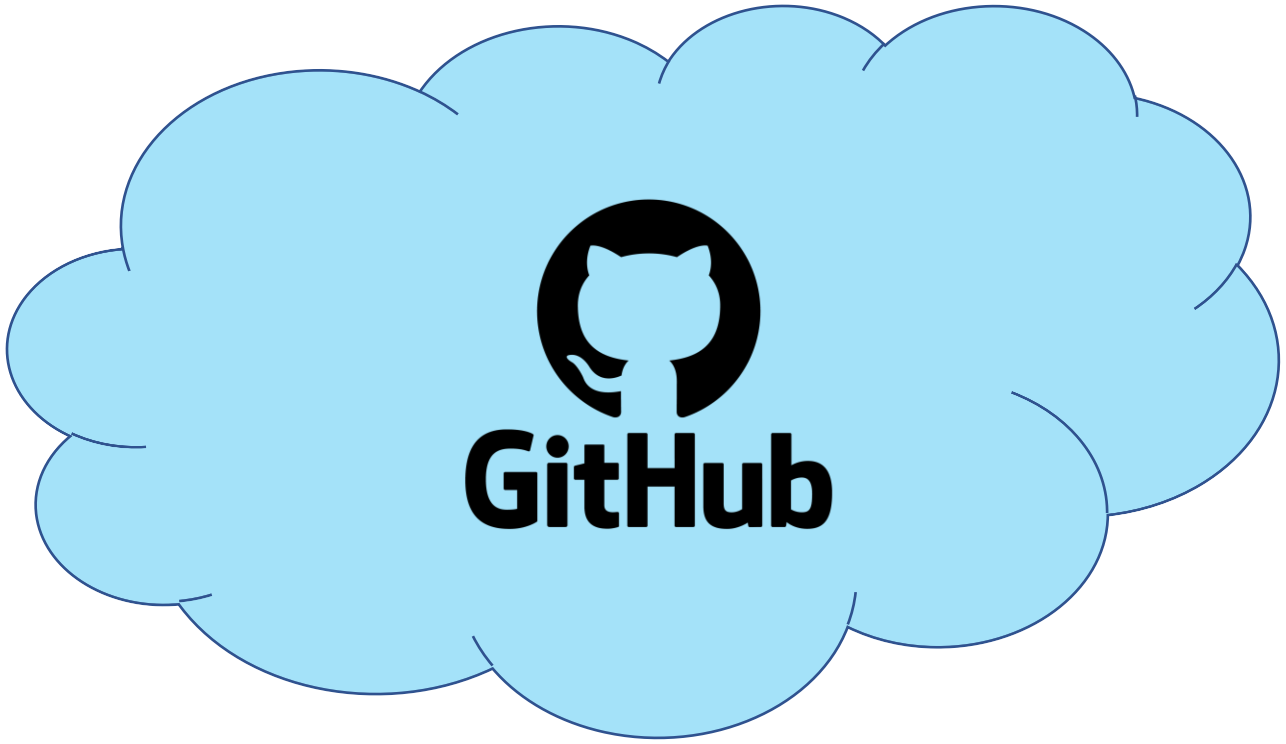
GitHub is the single largest host for Git repositories, and is the central point of collaboration for millions of developers and projects.

- The first thing you need to do is set up a free user account.
- Simply visit https://github.com, choose a user name that isn't already taken, provide an email address and a password, and click the big green "Sign up for GitHub" button.

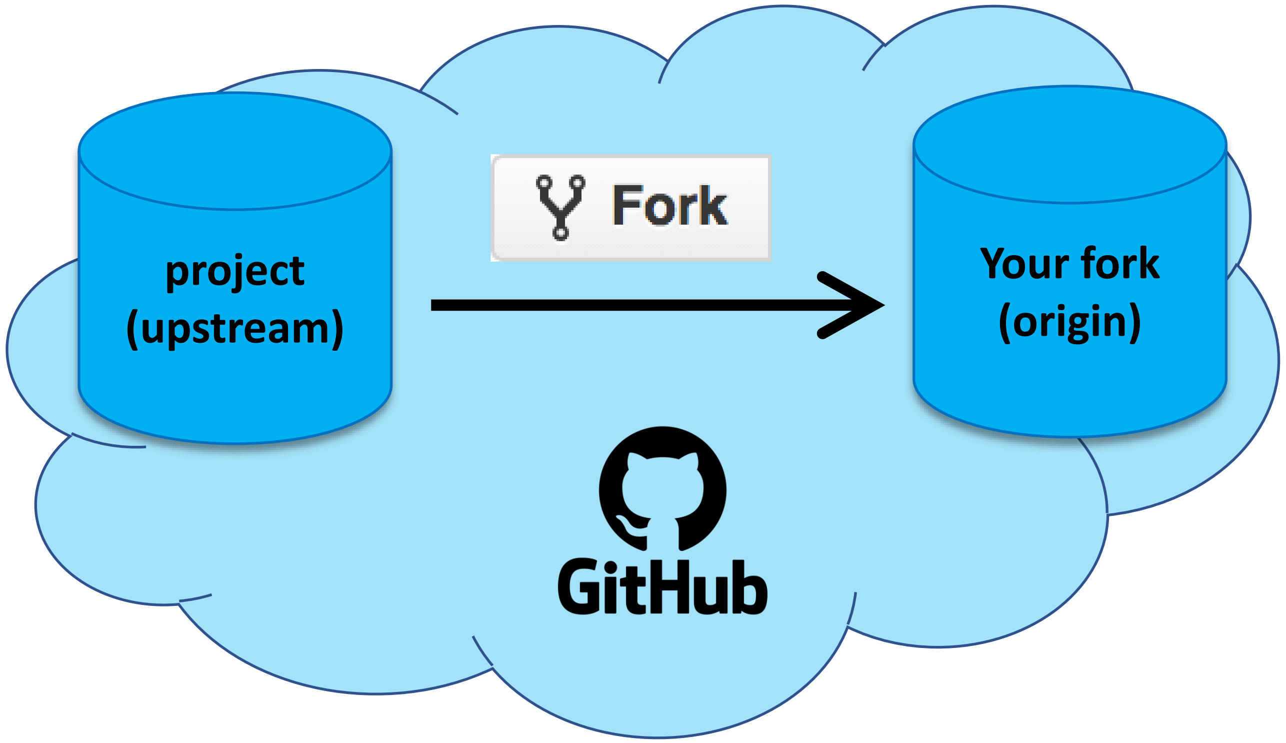


```
Welcome to GitHub!
Let's begin the adventure

Enter your email
✓ jamil@sejong.ac.kr

Create a password
✓ ••••••••••

Enter a username
✓ Jamil-Hussn

Would you like to receive product updates and announcements via
email?
Type "y" for yes or "n" for no
→
```

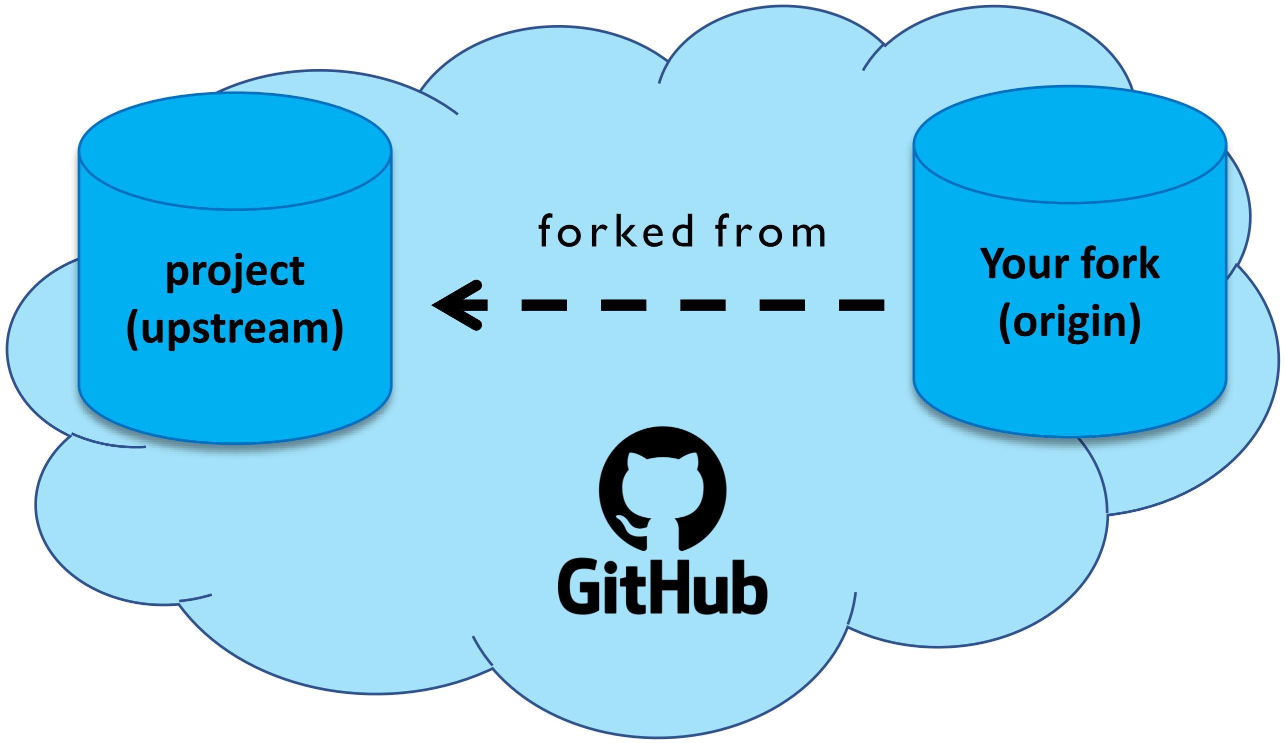
Contributing to a Project



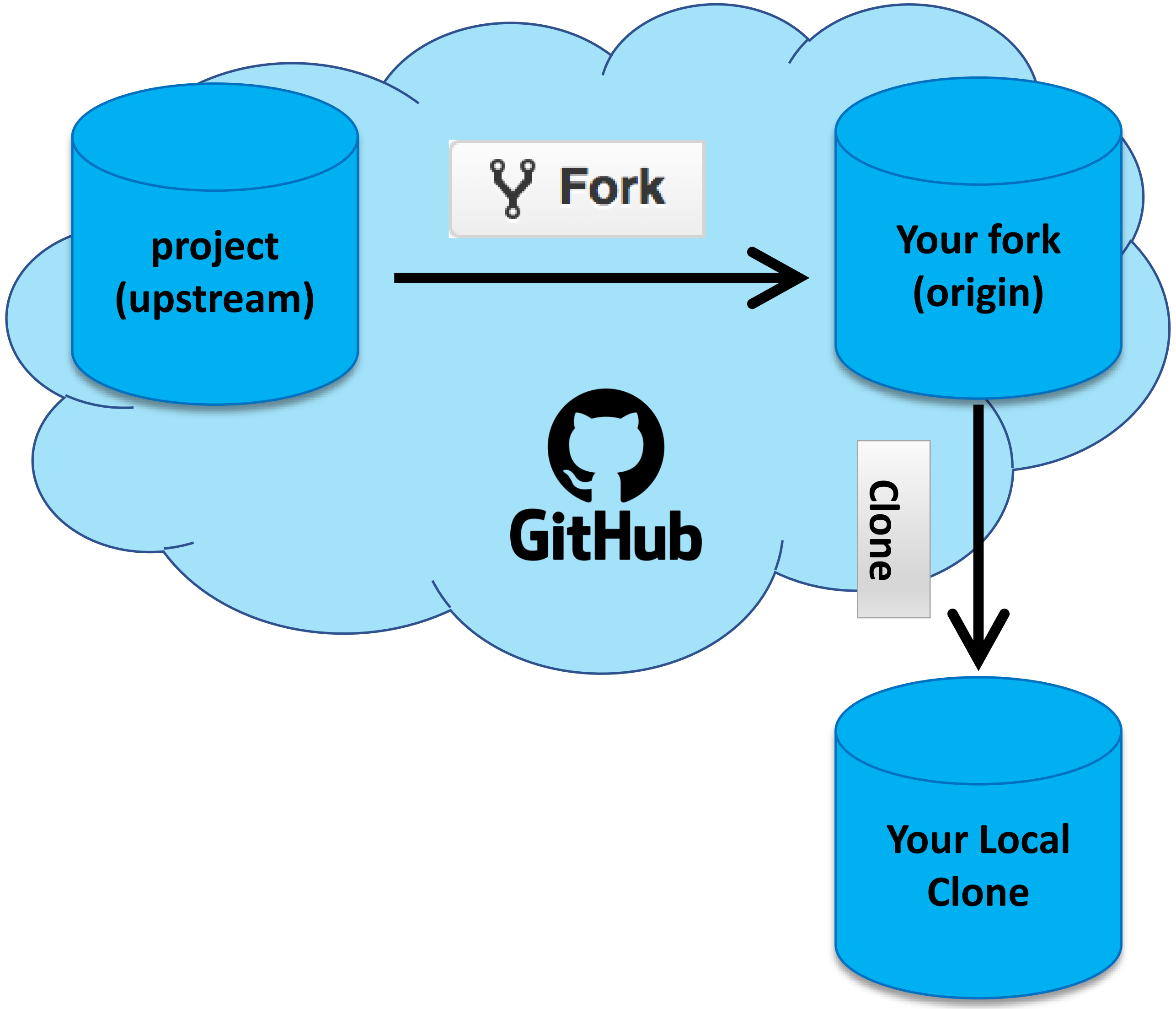
Contributing to a Project



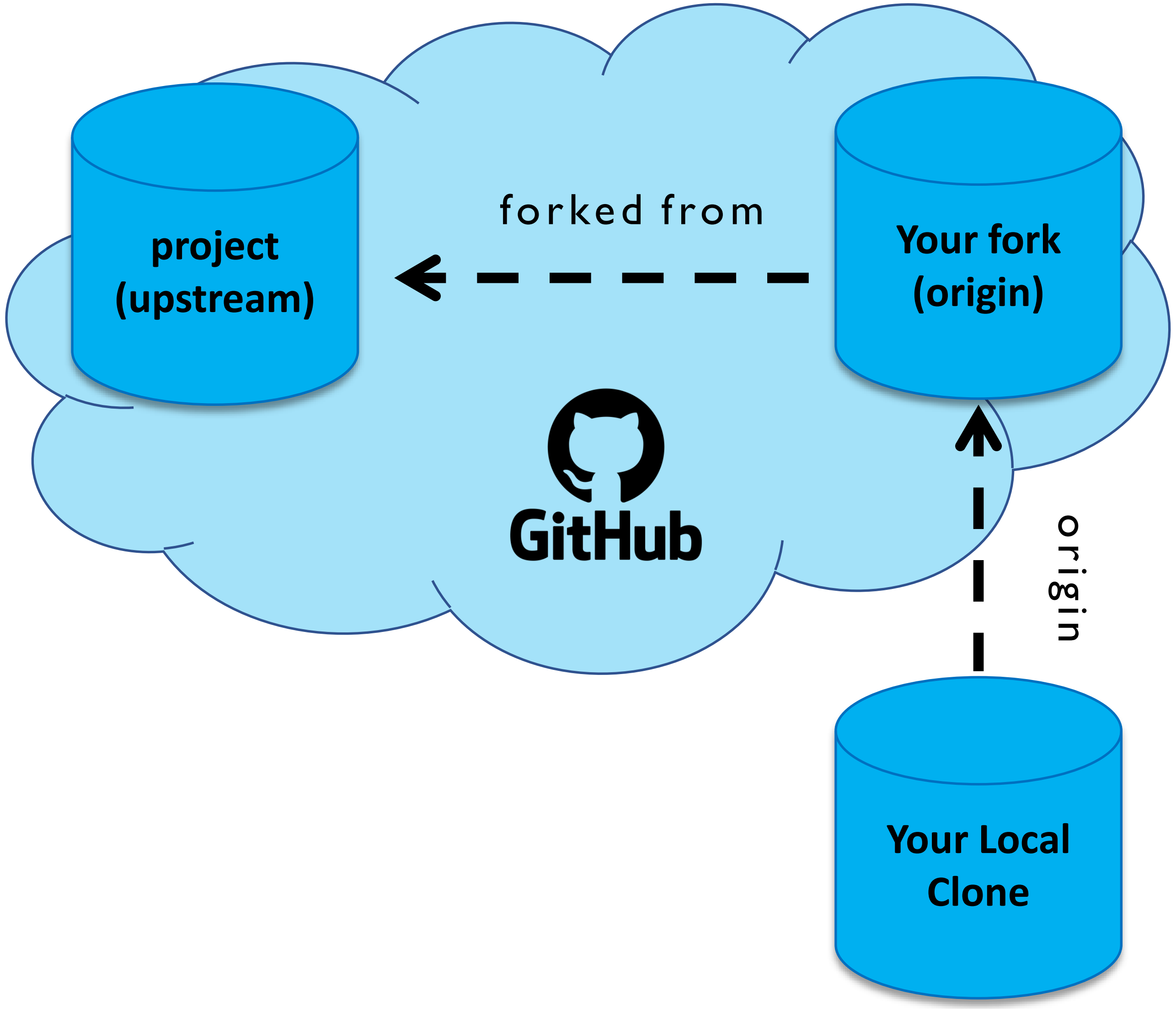
Contributing to a Project



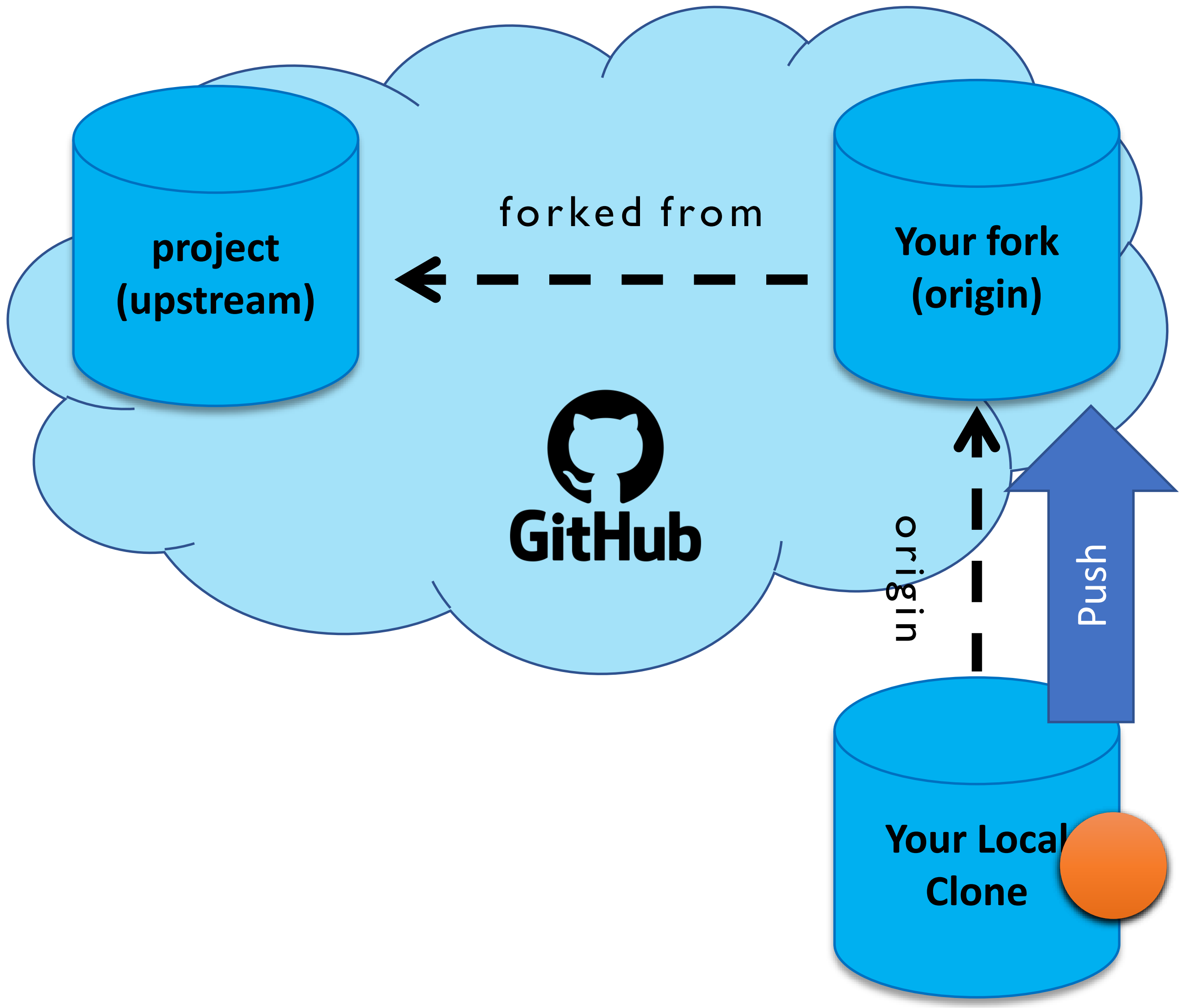
Contributing to a Project



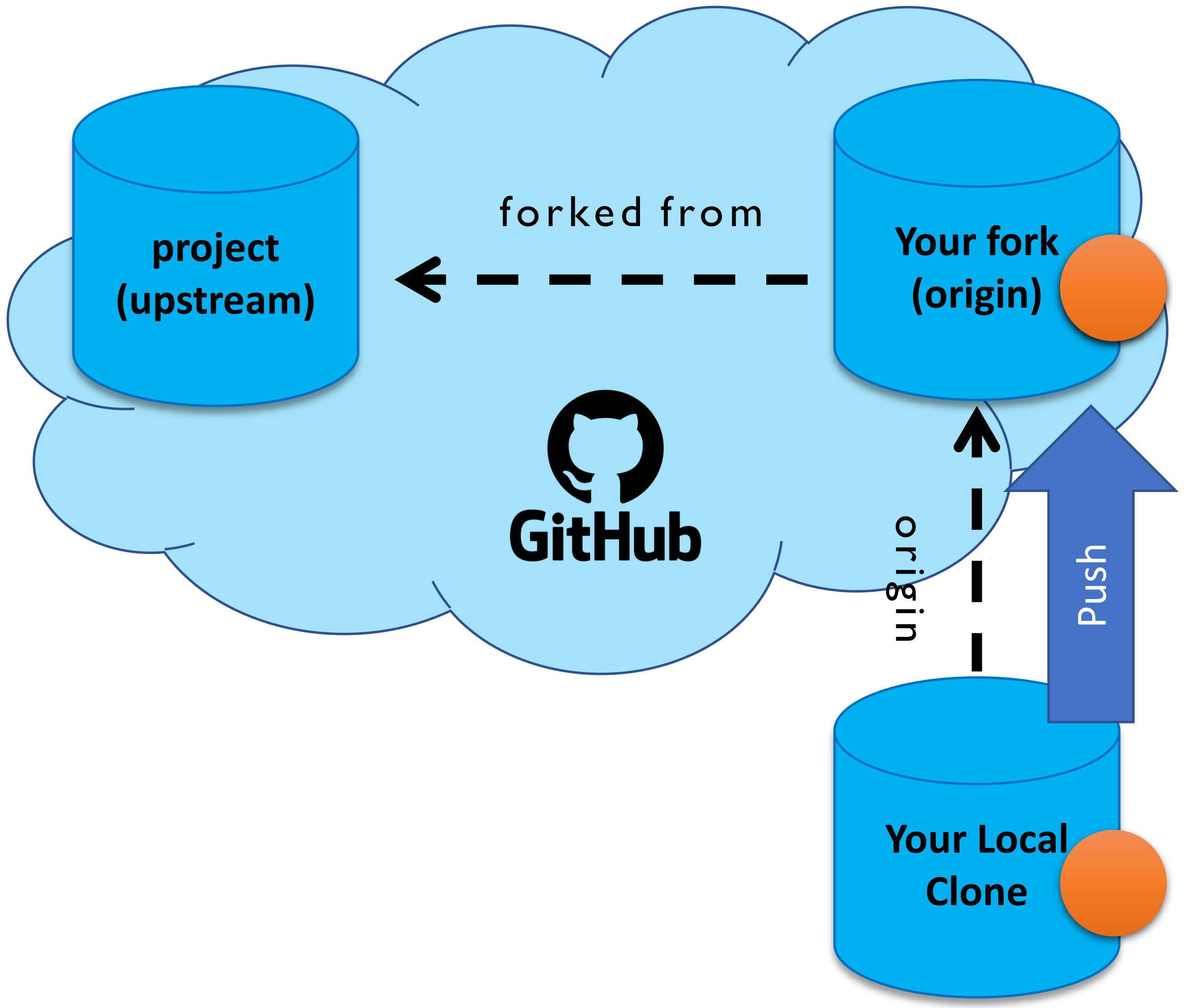
Contributing to a Project



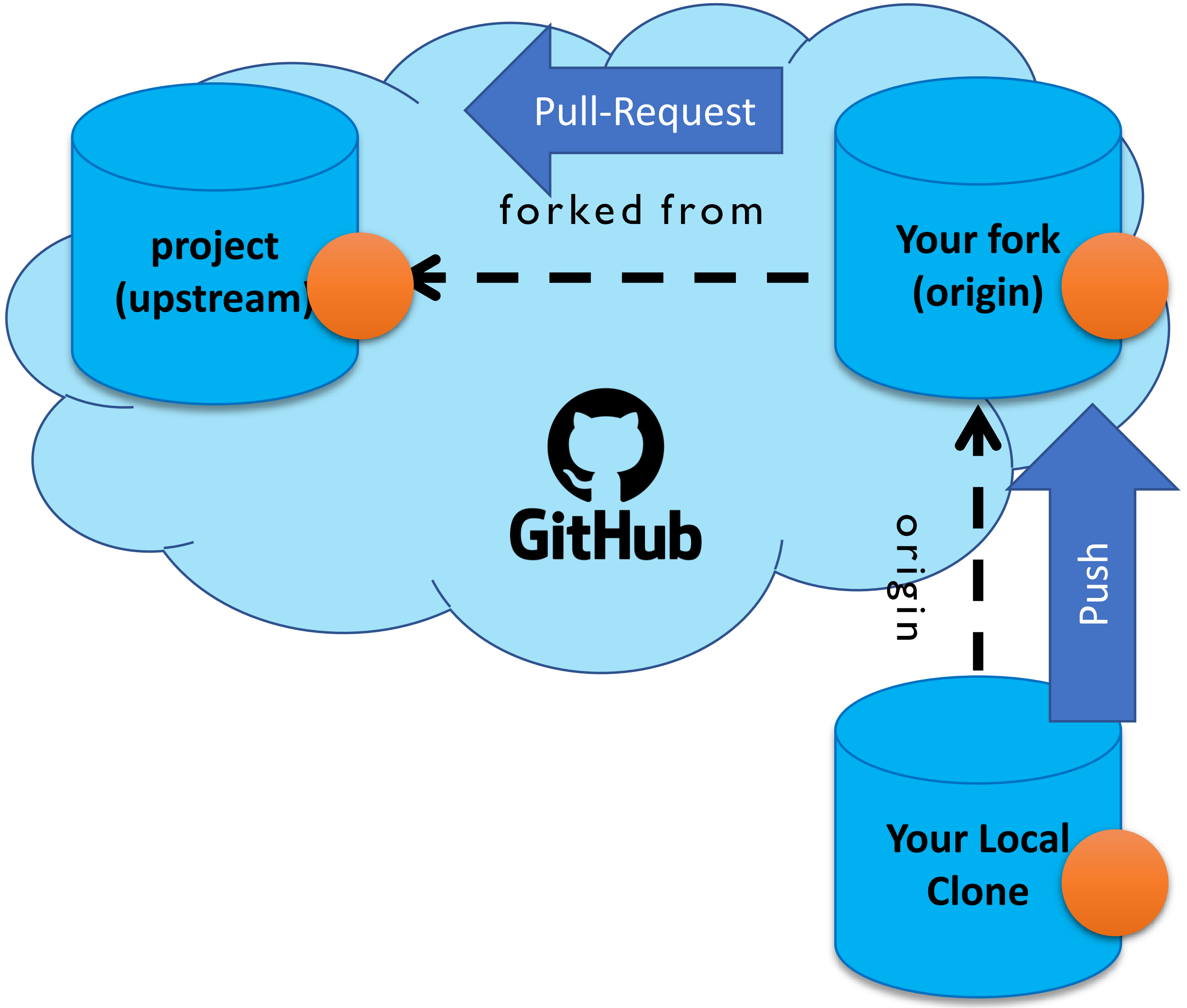
Contributing to a Project



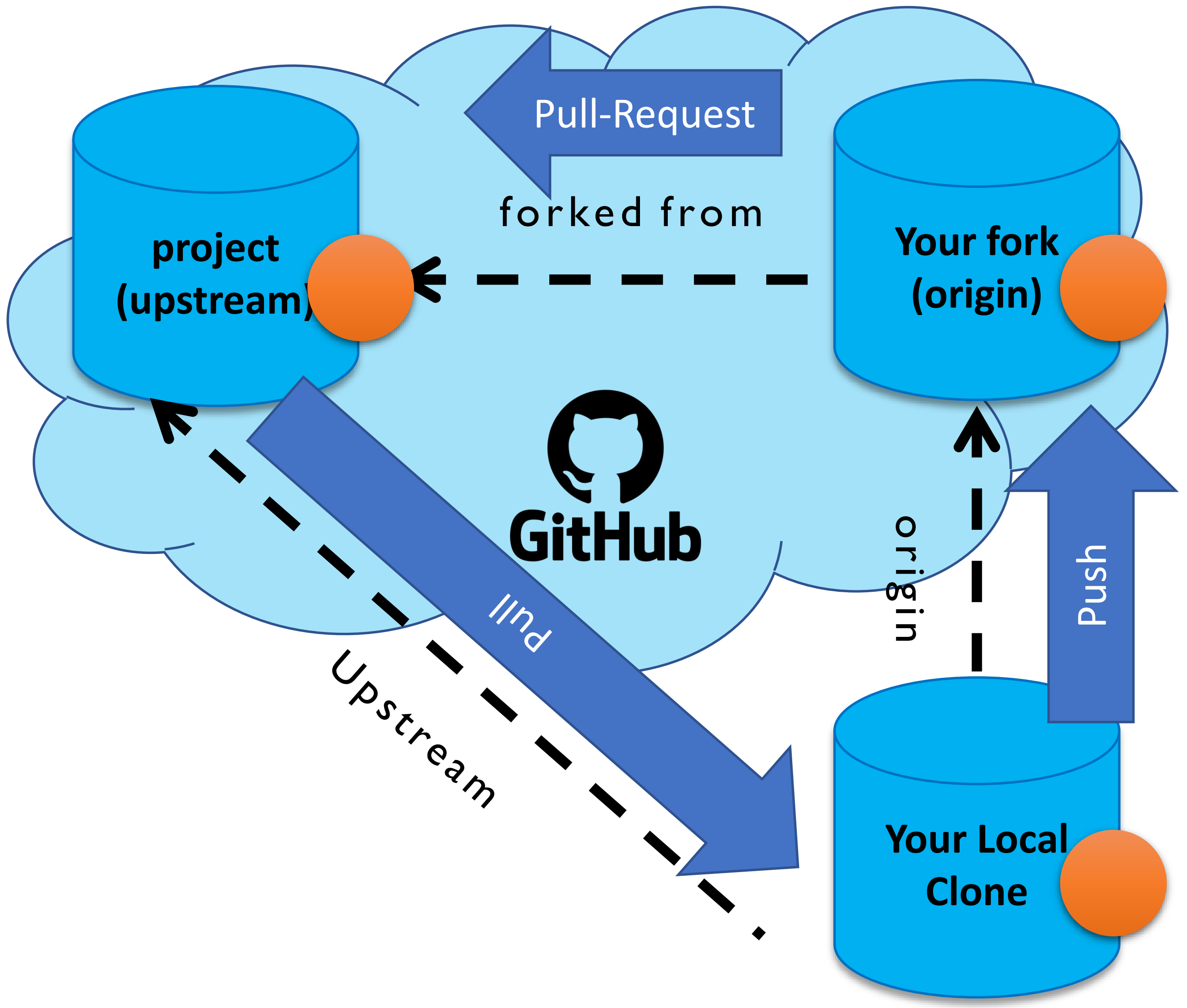
Contributing to a Project



Contributing to a Project



Contributing to a Project



Working with Remotes

- To be able to collaborate on any Git project, you need to know how to manage your remote repositories.
- To add a new remote Git repository as a shortname you can reference easily, run git remote add <shortname> <url>:

```
$ git clone https://github.com/schacon/simplegit-progit
```

```
$ git remote -v
```

```
$ git remote add pb https://github.com/paulboone/ticgit
```

```
$ git fetch pb
```

세종대학교
SEJONG UNIVERSITY

Fetching and Pulling from Your Remotes

- As you just saw, to get data from your remote projects, you can run:

```
$ git fetch <remote>
```

- If you clone a repository, the command automatically adds that remote repository under the name “origin”.

```
$ git fetch origin
```

Fetching and Pulling from Your Remotes

- git pull is a command used in Git to fetch changes from a remote repository and automatically merge them into the current branch.

```
git pull = git fetch + git merge
```

```
$ git pull origin main
```

Pushing to Your Remotes

- When you have your project at a point that you want to share, you have to push it upstream
- The command for this is simple: git push <remote> <branch>

```
$ git push origin master
```

Tagging – 1/4

- Like most VCSs, Git has the ability to tag specific points in a repository's history as being important.
- Typically, people use this functionality to mark release points (v1.0, v2.0 and so on).
- Listing Your Tags

```
$ git tag
```

```
$ git tag -l
```

```
$ git tag -l "v1.8.5*"
```

Tagging - Creating Tags – 2/4

- **Git supports two types of tags:**
  - Annotated
  - Lightweight
- **Annotated Tags**
  - Creating an annotated tag in Git is simple. The easiest way is to specify -a when you run the tag command, -m specifies a tagging message, which is stored with the tag

```
$ git tag -a v1.4 -m "my version 1.4"
```

- **Lightweight Tags**
  - To create a lightweight tag, don’t supply any of the -a, -s, or -m options, just provide a tag name:

```
$ git tag v1.4-lw
```

Tagging - Sharing Tags – 3/4

- By default, the git push command doesn’t transfer tags to remote servers.
- You will have to explicitly push tags to a shared server after you have created them.
- This process is just like sharing remote branches — you can **run git push origin <tagname>**.
- **If you have a lot of tags that you want to push up at once, you can also use the --tags option to the git push command.**

```
$ git push origin v1.5
```

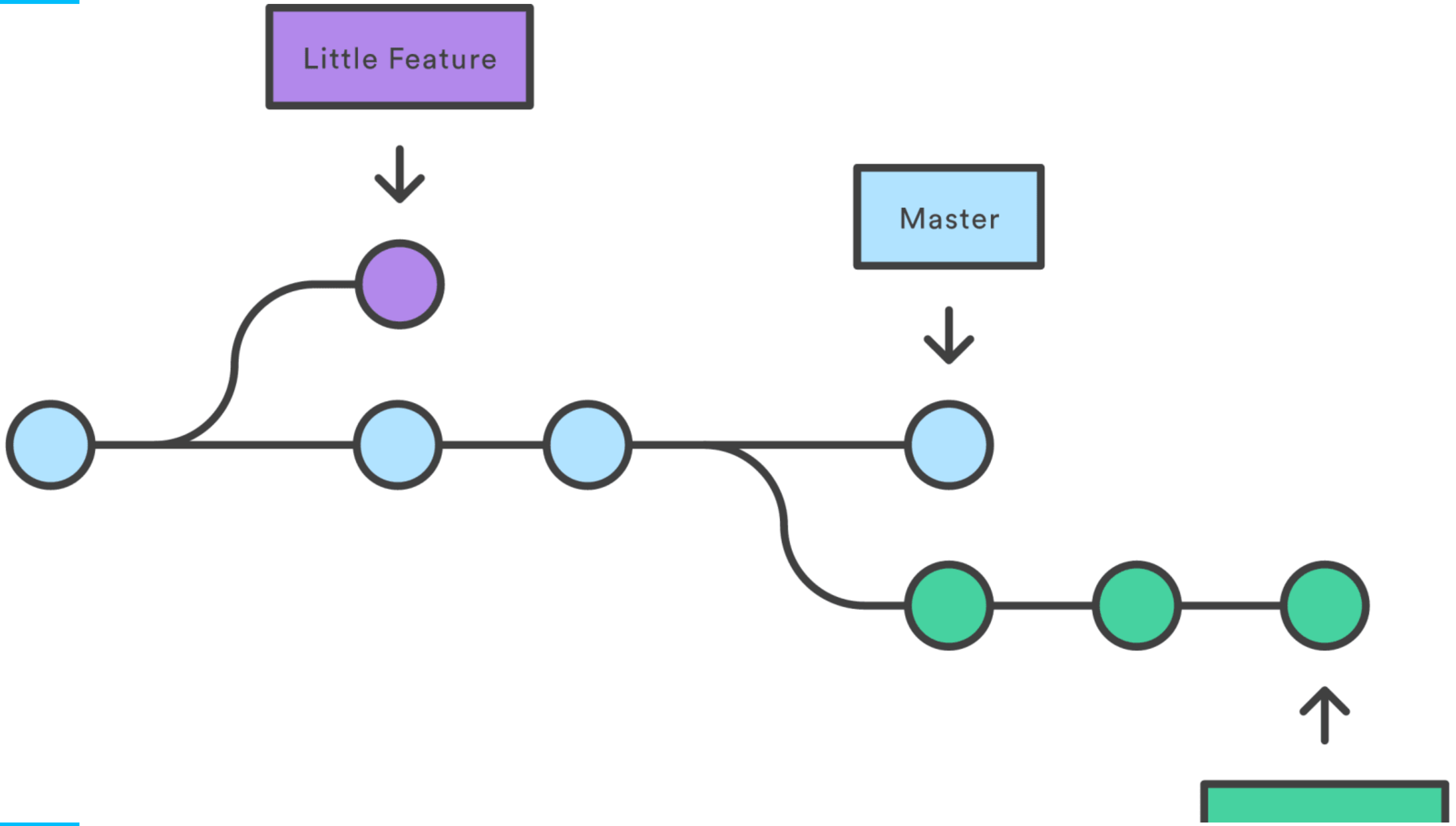
```
$ git push origin --tags
```

Tagging - Deleting Tags – 4/4

- To delete a tag on your local repository, you can use **git tag -d <tagname>**
- For example, we could remove our lightweight tag above as follows:

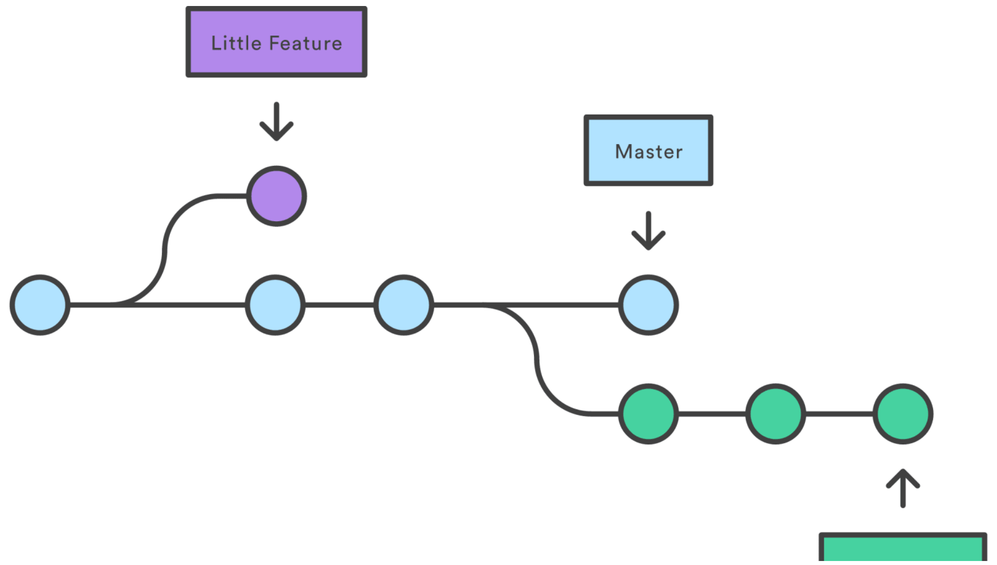
```
$ git tag -d v1.4-lw
```

Branches



Git Branching

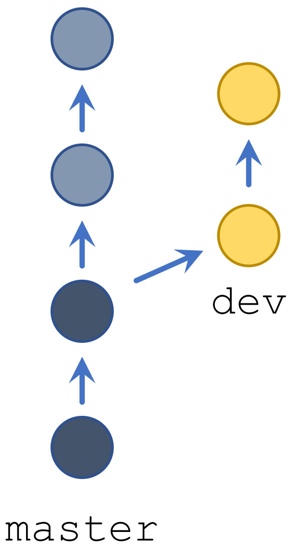
- Branching means you diverge from the main line of development and continue to do work without messing with that main line
- The way Git branches is incredibly lightweight, making branching operations nearly instantaneous, and switching back and forth between branches generally just as fast.



Branching and merging with `git`

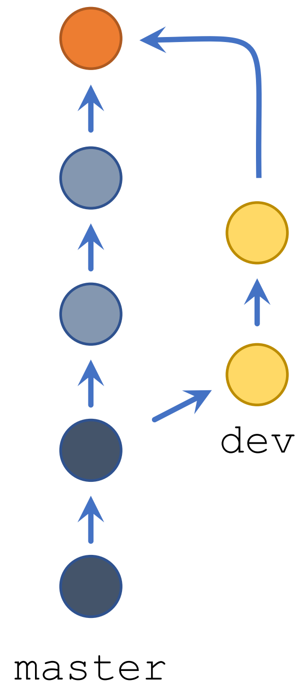
- There are multiple methods to bring parallel developments back together

Getting started with branching

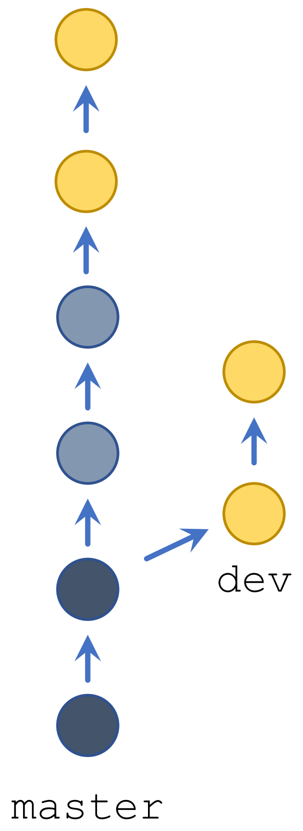


Three options to ***merge*** the changes from `dev` into `master`

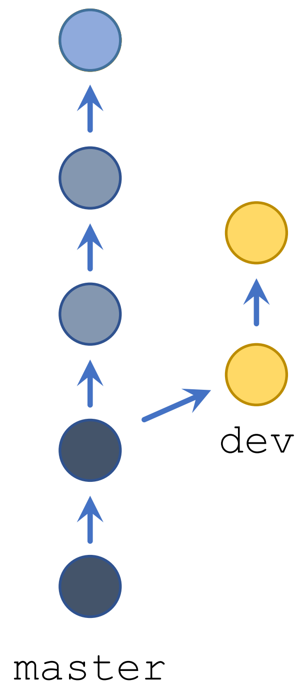
1) A merge commit



2) Rebase

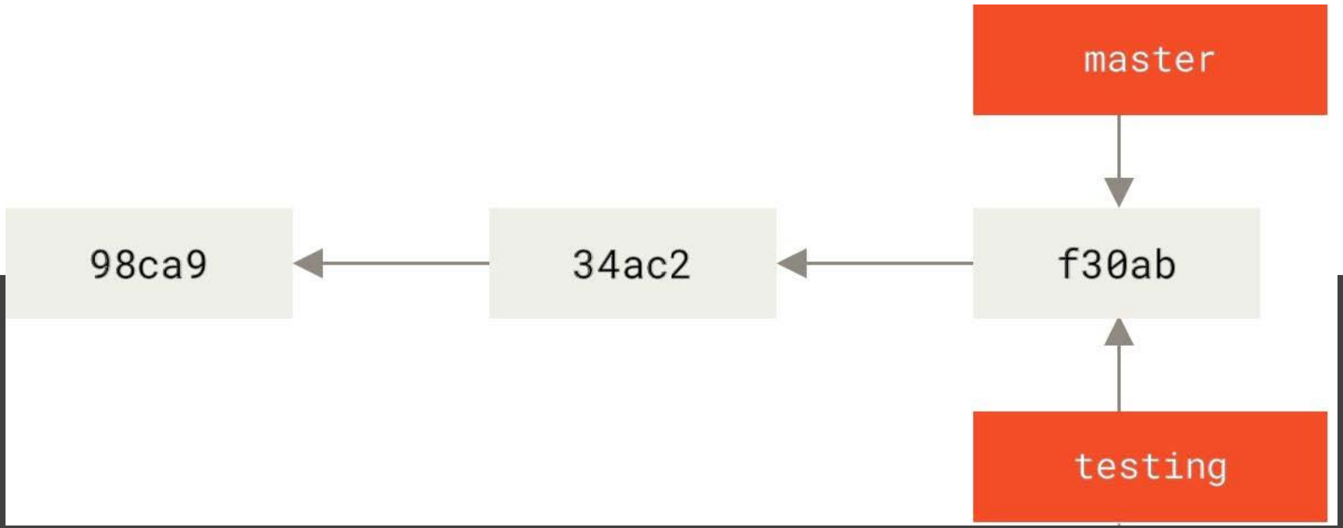


3) Squash and merge



Creating a New Branch

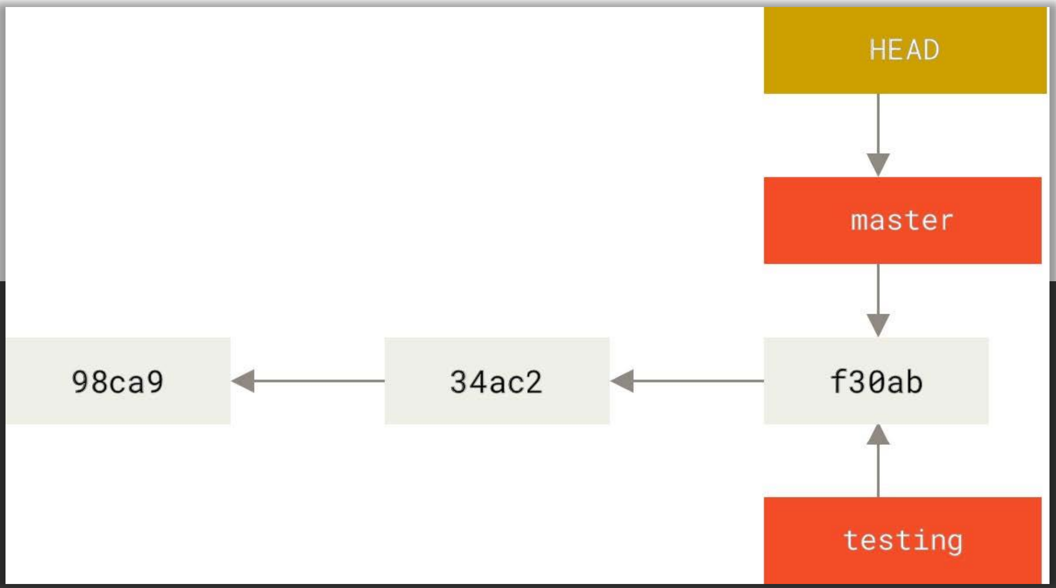
- What happens when you create a new branch? Well, doing so creates a new pointer for you to move around.
- Let’s say you want to create a new branch called testing. You do this with the git branch command:

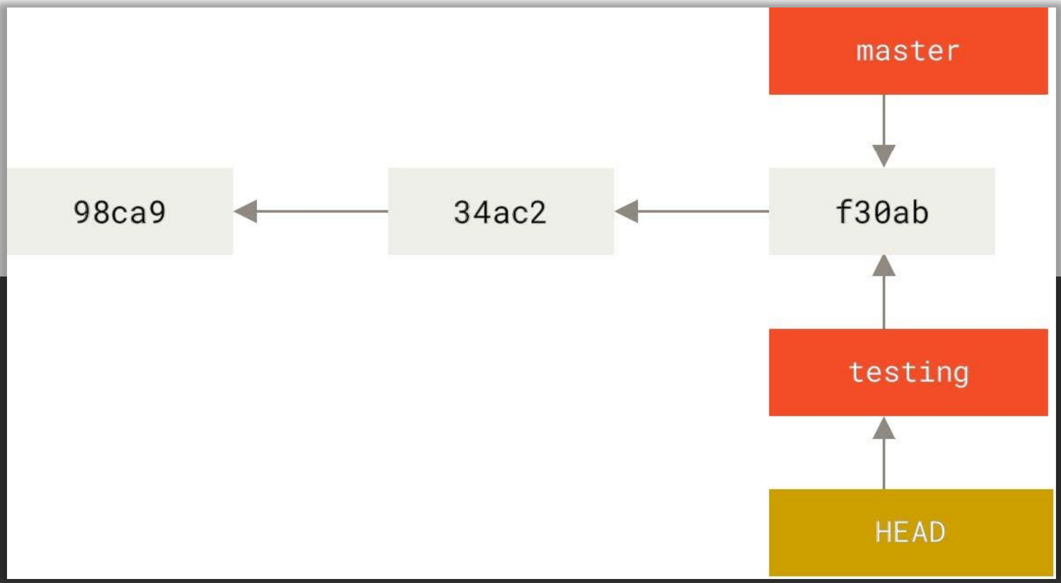


```
$ git branch testing
```

Switching Branches – 1/2

- To switch to an existing branch, you run the git checkout command.
- Let’s switch to the new testing branch:





```
$ git checkout testing
```

Switching Branches – 2/2

- To create a new branch and switch to it at the same time, you can run the git checkout command with the -b switch:

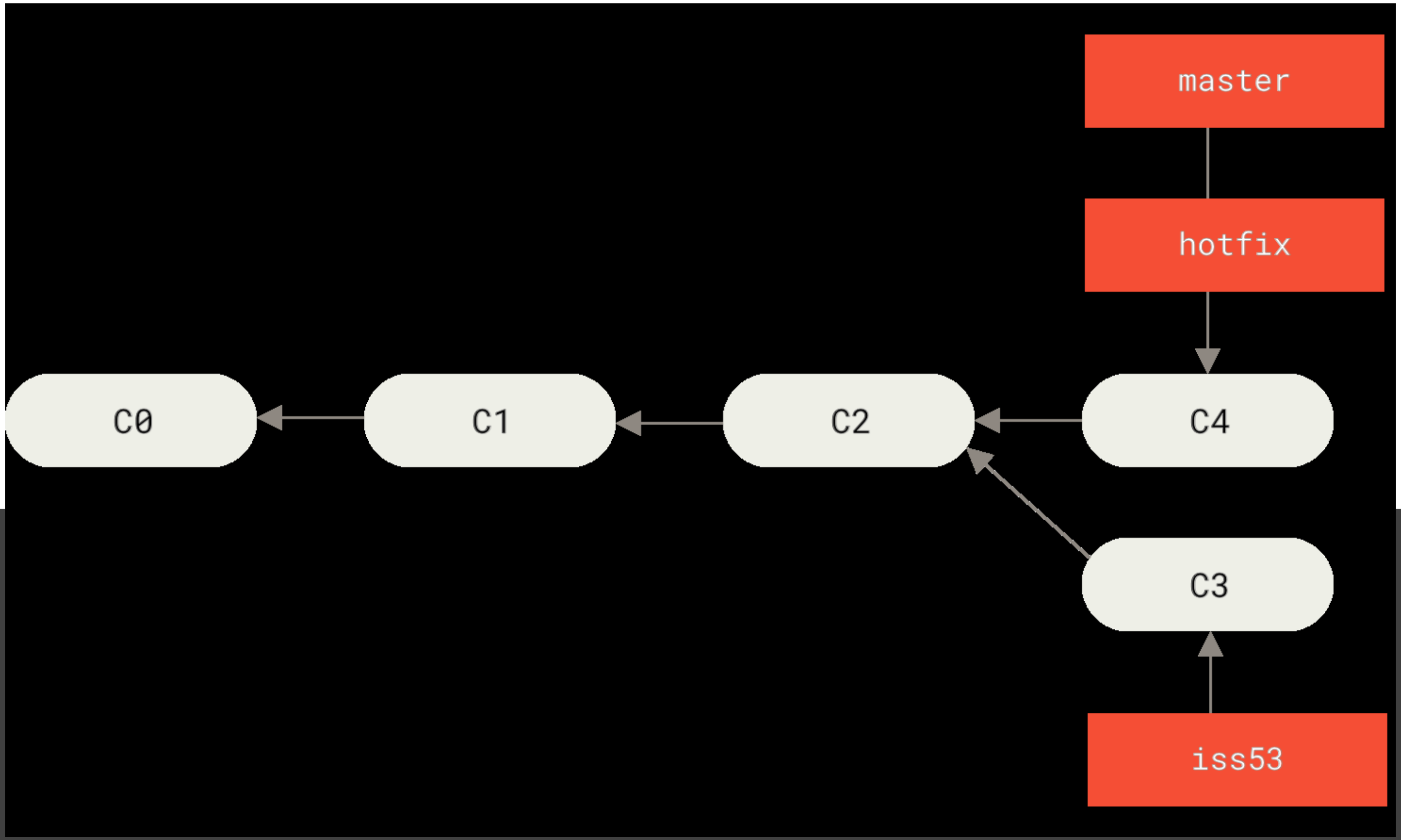
```
$ git checkout -b iss53
```

- This is shorthand for:

```
$ git branch iss53
$ git checkout iss53
```

Delete Branches

- you can delete it with the -d option to git branch:



```
$ git branch -d hotfix
```

Basic Merging

- Suppose you’ve decided that your issue #53 work is complete and ready to be merged into your master branch.
- In order to do that, you’ll merge your iss53 branch into master, much like you merged your hotfix branch earlier.
- All you have to do is check out the branch you wish to merge into and then run the git merge command:

```
$ git checkout master'
$ git merge iss53
```

Reading Materials

- Book : Pro Git Scott Chacon, Ben Straub
- https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository

Thanks

Office Time: Monday-Friday (1000 - 1800)

You can send me an email for meeting, or any sort of discussion related to class matters.

jamil@sejong.ac.kr

세종대학교
SEJONG UNIVERSITY