NUS HACKERS

PRESENTS

Introduction to Git (and GitHub)

17 May 2025

References

These slides are complemented by the wiki so please refer to it for more details/information

Wiki
Slides

Things you need

git

GitHub

Setup Issues

Quick check: Did anyone face any problems setting up Git on your machine or creating an account on GitHub?

If you have any issues, refer to the setup guide provided in the wiki.

Config

Accessing GitHub

Using your GitHub password is no longer allowed after **August 13, 2021**. If you use HTTPS, you'll need to create a Personal Access Token (PAT).

In your GitHub account, go to `Settings > Developer Settings > Personal Access Token` to generate one.

Fundamental Concepts

What is Git?

- Distributed version control system (DVCS)
- Created by Linus Torvalds in 2005 as a replacement for existing VCS for the Linux kernel

What is **version control**?

- A **version control system (VCS)** helps to record changes to a file or a set of files over time
- It allows you to revert a project to a previous state, or to compare changes over time
- Acts as an "undo" button for your code and lets your collaborate with your team

Local vs remote repositories

- **Repository:** parent folder that Git monitors for changes
- **Local:** exists on local machine
- **Remote:** exists on external server such as Github

Thinking in commits

- Snapshot of the instance of the codebase at a given point in time
- Incrementally added over past snapshots as a set of changes
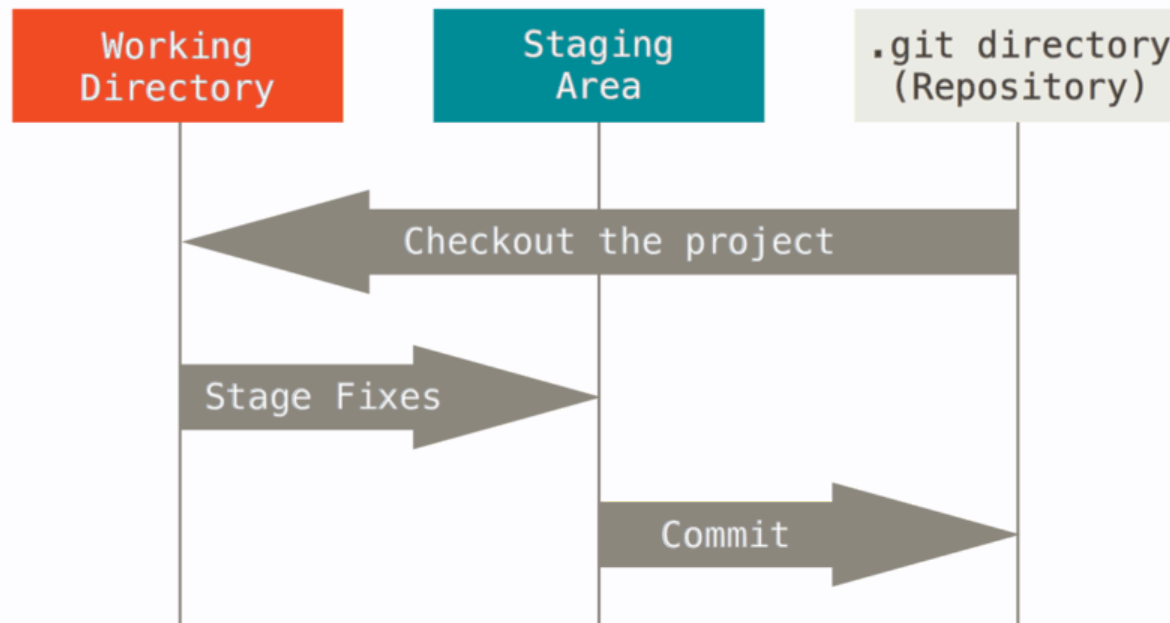- Visualized as a Directed Acyclic Graph (DAG)
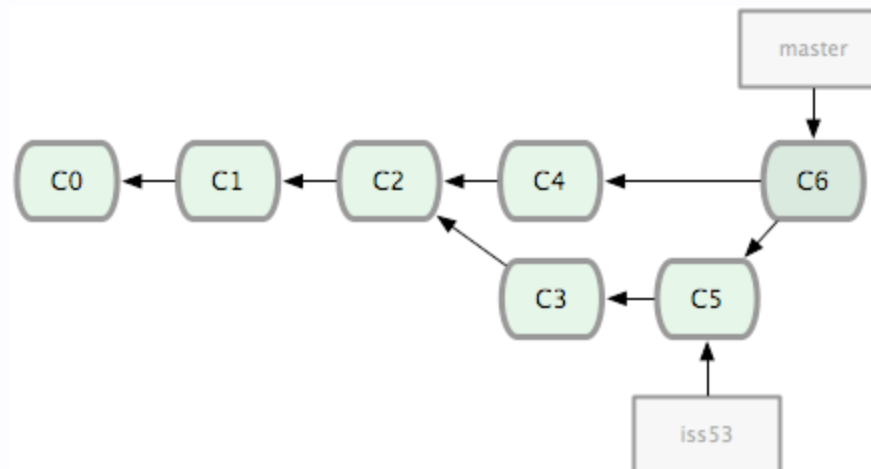
Branching away

- Independent line of work that stems/branches off from a point in development
- Useful for isolating bug fixes/features
- Default branch: `main`

Adding files to a snapshot

- Working directory: where you actually work
- Index/"staging" area: where you construct a commit
- Repository/commit: the repository itself

Getting Started

Creating a local repository

```
mkdir new-folder/
cd new-folder/
git init
```

Adding a file

```
echo 'Hello world' >> hello.txt
```

Adding `hello.txt` to your snapshot

```
git add hello.txt
```

Check status

```
git status
```

Taking the snapshot

```
git commit -m "First commit"
```

Check status

```
git status
git log
```

Integrating Remote Repositories

Creating a new GitHub repository

Go to https://github.com/new

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner *   **Repository name ***

woojiahao ▾   /   new-folder

✓ **new-folder is available.**

Great repository names are short and memorable. Need inspiration? How about **musical-octo-lamp** ?

Description (optional)

Something descriptive

○ 🖥 **Public**
    Anyone on the internet can see this repository. You choose who can commit.

○ 🔒 **Private**
    You choose who can see and commit to this repository.

Initialize this repository with:

☐ **Add a README file**
    This is where you can write a long description for your project. Learn more about READMEs.

Add .gitignore

.gitignore template: None ▾

Choose which files not to track from a list of templates. Learn more about ignoring files.

Choose a license

License: None ▾

A license tells others what they can and can't do with your code. Learn more about licenses.

ⓘ You are creating a public repository in your personal account.

Create repository

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

new-folder Public

Pin Unwatch 1 ▾ Fork 0 ▾ Star 0 ▾

Set up GitHub Copilot
Use GitHub's AI pair programmer to autocomplete suggestions as you code.

Get started with GitHub Copilot

Add collaborators to this repository
Search for people using their GitHub username or email address.

Invite collaborators

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

Set up in Desktop or HTTPS SSH git@github.com:woojiahao/new-folder.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 "# new-folder" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:woojiahao/new-folder.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin git@github.com:woojiahao/new-folder.git
git branch -M main
git push -u origin main
```

💡 **ProTip!** Use the URL for this page when adding GitHub as a remote.

Connecting local repository to remote repository

`origin` is used as the name for the remote (it's just convention).

```
git remote add origin git@github.com:<github username>/<repository name>.git
git branch -M main
```

Uploading local repository snapshots

```
git push -u orign main
```

Receiving remote repository snapshots

```
git pull origin main
```

Cloning remote repositories

Downloading a local copy of the repository

```
cd ../
git clone git@github.com:<github username>/<repository name>.git another-folder/
```

Branching

What is a branch?

A branch is an independent line of development, often used for features or bug fixes
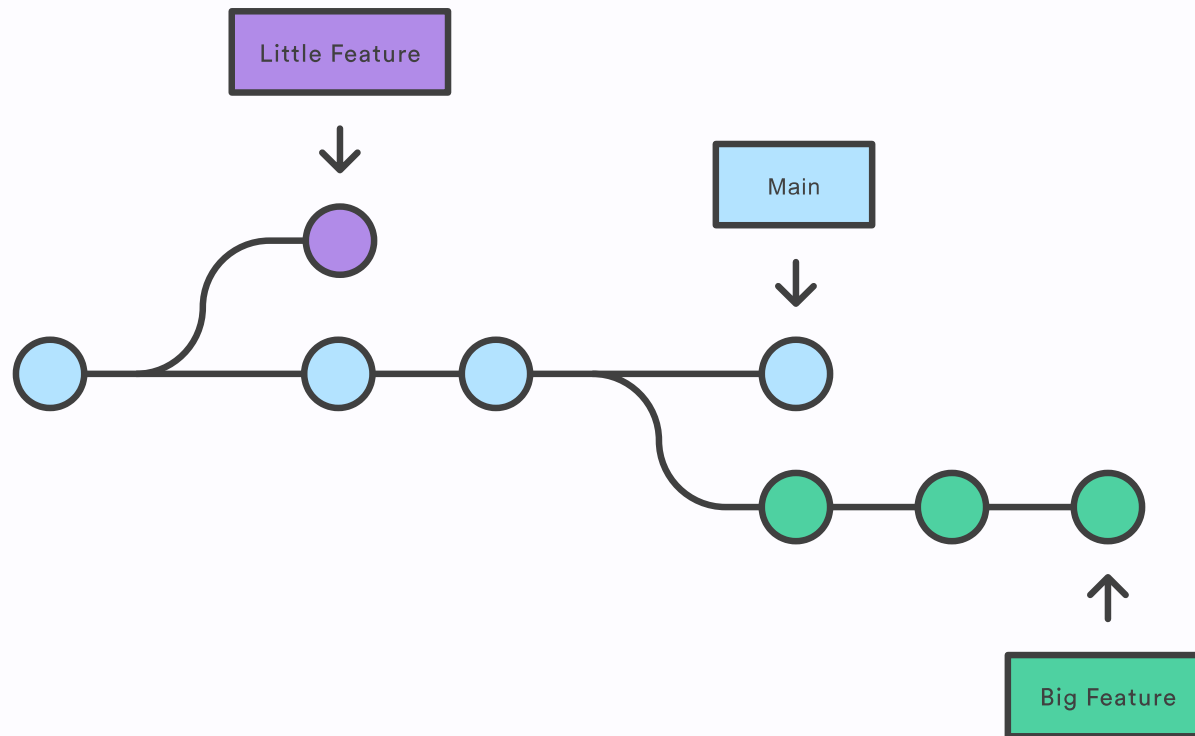
HEAD

HEAD is a special name given to the current commit
of your current branch for ease of reference
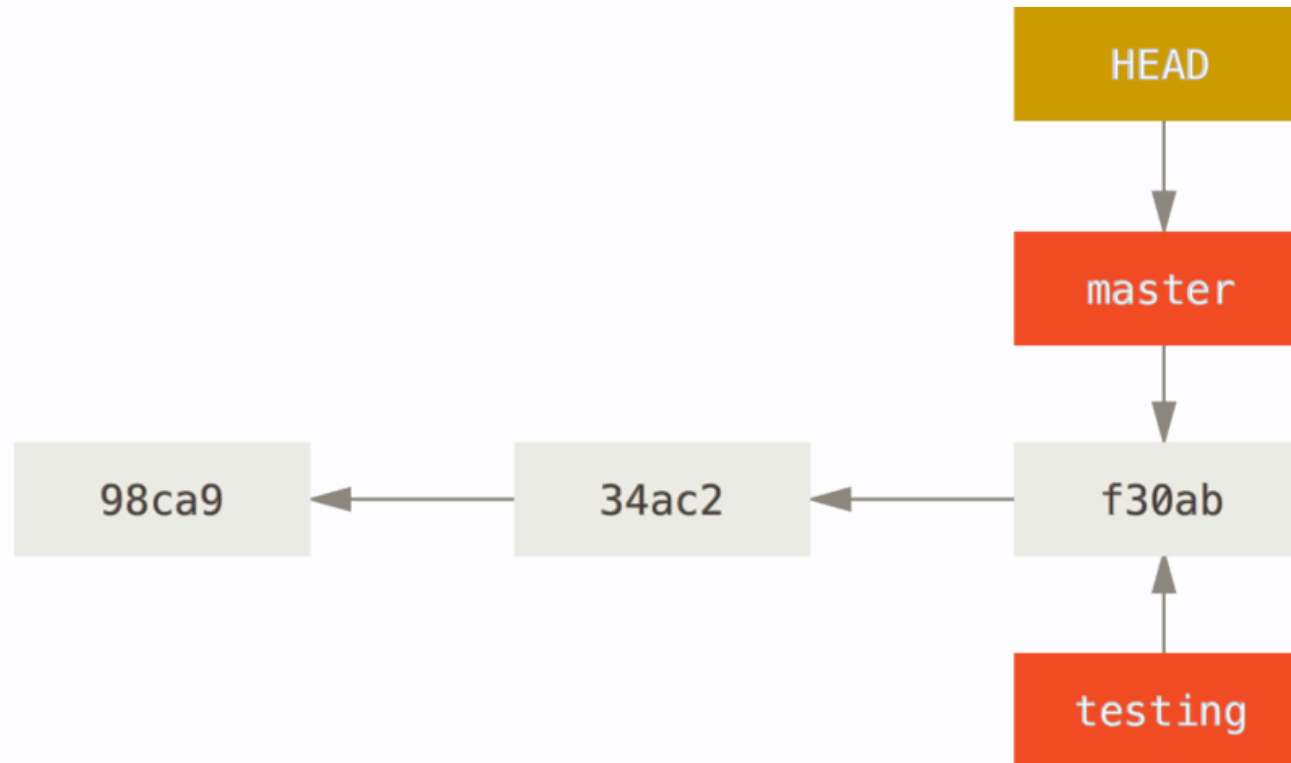
Creating a branch

```
git checkout -b new-feature
```

Alternatively:

```
git branch new-feature
git checkout new-feature
```

Changing branches

```
git checkout main
```

Viewing all branches

```
git branch -v
```

Deleting a branch

```
git branch -d <branch name>
```

Renaming a branch

```
git branch -m <new branch name>
```

Combining changes of branches

main is the target branch, feature-A is the source branch

```
git checkout main
git merge feature-A
```

Merge Conflicts

Merge conflicts

Occur when two (or more) modifications are made to the same line of a file; creating a state of confusion for Git
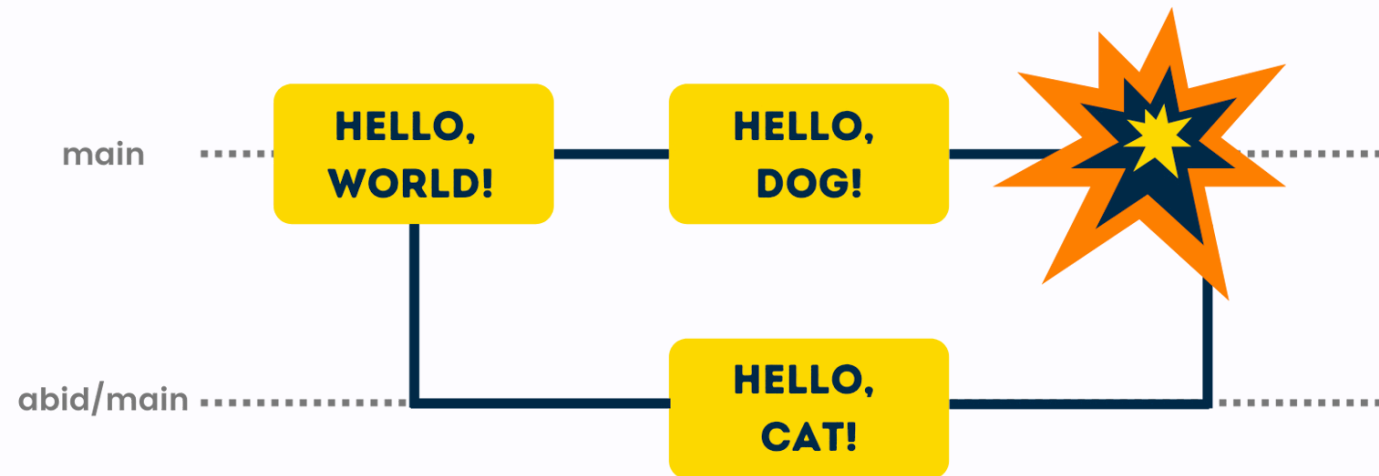
Simulating merge conflicts

In your repository, create branch `conflict-1` from `main` and edit the first line of `hello.txt`

```
git checkout -b conflict-1
```

Return to `main` and create another branch `conflict-2` from `main` and edit the first line of `hello.txt` again

```
git checkout main
git checkout -b conflict-2
```

Merge conflicts

Now, try to merge `conflict-1` and `conflict-2` into `main` .

```
git merge conflict-1
```

```
Updating b53e9cf..87a92c3
Fast-forward
    hello.txt | 2 +-
    1 file changed, 1 insertion(+), 1 deletion(-)
```

Merge conflicts

`git merge conflict-2`

```
Auto-merging hello.txt
CONFLICT (content): Merge conflict in hello.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Handling merge conflicts

```
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)

You have unmerged paths.
(fix conflicts and run "git commit")
(use "git merge --abort" to abort the merge)

Unmerged paths:
(use "git add <file>..." to mark resolution)

both modified:    hello.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Handling merge conflicts

```
$ cat hello.txt
<<<<<<< HEAD
Goodbye!
=======
Farewell!
>>>>>>> conflict-2
```

Top half: current content in branch

Bottom half: content that is about to be merged

Handling merge conflicts

Edit `hello.txt` as such...

Completing the process

Commit the conflicting file

```
git commit
```

Intermission

5 minutes...

In the meantime, try out the commands that you have learnt!

Add new files, edit them, commit them, and stage them!

Collaborative Workflows

Common Workflows

1. Fork & PR
2. Branch & PR

This workshop focuses on the former - "Fork & PR"

The wiki contains more information about Branch & PR

What is a PR?

Pull requests are like a Request for Comments for a set of changes made on a separate branch

Allows developers and contributors to share their comments on the changes

Creating a PR

Push a local branch to a remote repository

```
git checkout -b sample-pr
vim hello.txt
git add hello.txt
git commit -m "new changes"
git push origin sample-pr
```

Visit GitHub to create the PR (screenshots on the wiki)

Fork & PR workflow

Forking: creating an at-the-time copy of a remote repository (only needs to be done once per remote repository)

Upstream: common name for the original repository

Branch & PR Workflow

Each member owns their copy of the repository locally (through cloning)

When working on a new feature or bug fix, each member will

Branch & PR Workflow

1. Pull the latest changes from the remote repository
2. Create a branch per feature/bug fix on their local copy
3. Edit the files in their respective branch
4. Push their local branch to the repository
5. Make a pull request of their feature/bug fix branch to the `main` branch (remote copy)

Hands-on time!

1. Create a fork of the demo repository
2. Clone the fork to local machine
3. Make changes to the repository
4. Commit changes
5. Push changes to forked repository
6. Create PR through Github

Intermission!

5 minute break and Q&A!

Additional concepts

Commit manipulation

Revert

Undo accidental changes made

```
$ git log --graph --oneline
* d1f4fcc (HEAD -> master, origin/master, origin/HEAD) Add file3
* 643aec6 Update file to c
* 4ec21c7 Update file to b
* 055cab4 Initial commit
```

Suppose we want to revert to commit `643aec6`.

Revert

```
$ git revert 643aec6
[master 7b73baf] Revert "Update file to c"
    1 file changed, 1 insertion(+), 1 deletion(-)
$ git show
commit 7b73baf229e2b8db19bc594c450743b50adf649d (HEAD -> master)
Author: Your Name <your@email.com>
Date:   Tue May 11 01:21:31 2021 +0800

    Revert "Update file to c"

    This reverts commit 643aec6d2a1b4cd485d678886fc1cef25b15bee0.

diff --git a/file b/file
index f2ad6c7..6178079 100644
--- a/file
+++ b/file
@@ -1 +1 @@
-c
+b
```

Reset

Undo `git add`.

```
$ git status
On branch master
Changes to be committed:
    (use "git restore --staged <file>..." to unstage)
    modified:   file
$ git reset file
Unstaged changes after reset:
M   file
$ git status
On branch master
Changes not staged for commit:
    (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
    modified:   file
no changes added to commit (use "git add" and/or "git commit -a")
```

Restore

Undo changes to a file in the working tree.

```
$ echo e > file
$ git status
On branch master
Changes not staged for commit:
    (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
    modified:    file

no changes added to commit (use "git add" and/or "git commit -a")
$ git restore file
$ git status
On branch master
nothing to commit, working tree clean
```

Ignoring files

Sometimes we don't want Git to track a certain file

```
$ touch ignore-me
$ git status
```

```
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    ignore-me

nothing added to commit but untracked files present (use "git add" to track)
```

Ignoring files

We can add it to `.gitignore`

```
$ echo "/ignore-me" >> .gitignore
$ git status
```

```
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .gitignore

nothing added to commit but untracked files present (use "git add" to track)
```

`.gitignore` should be committed.

Viewing ignored files

```
$ git status --ignored
```

```
On branch master
Ignored files:
  (use "git add -f <file>..." to include in what will be committed)

      ignore-me

nothing to commit, working tree clean
```

What to ignore?

Typically, we ignore files like build artifacts and generated files that
are usually derived from the human-authored code in the repository. E.g.

- dependency caches like `/node_modules`
- compiled code like `.o` , `.pyc` files
- build output directories like `/bin` , `/out`
- runtime-generated files like log files
- personal configuration files e.g. of your IDE

.gitignore format

```
/logs/*/*.log
/logs/**/*.log
**/logs
**/logs/debug.log
*.log
/debug.log
debug.log
```

See the full pattern format.

Commit message discipline

First line: 80-character title, phrased imperatively

Then if your change is complex, elaborate on the change in prose.

```
Change greeting from "Hi" to "Hello"

"Hi" is a bit too informal for a greeting. We should change it to "Hello" instead,
so that our users don't feel like we are being too informal. Blah blah blah blah.
Blah blah.
```

Where to go from here?

Additional readings:

- Git manual
- Pro Git
- NUS Hackers Git Cheatsheet
- NUS Hackers Git Wiki
- Look into Gitworkflows
- Atlassian's collaboration guide

Where to go from here?

- If you're interested in how version control works with lots of technical details, look into Customizing Git and Git Internals
- GitHub isn't the only way you can share you repositories online! You could even self host your own Git servers.
- Why stop at learning? Build your own Git!

Where to go from here?

Check out Hackerschool: Advanced Git
(last run on 7 Nov 2020)

Recording: https://www.youtube.com/watch?v=pGAorBdZ6Y8

Slides: https://hs2010-git.github.io/adv

Where to go from here?

Check out Hackerschool: CI/CD with Github Actions (last run on 1 Apr 2025)

Recording: Coming Soon!

Wiki: https://wiki.nushackers.org/hackerschool/ci-cd-with-github-actions

End