UTHM
Universiti Tun Hussein Onn Malaysia

git Source Code Management System

Ts. Azamuddin bin Rasidi
Bahagian Teknologi Pendigitalan
Pusat Teknologi Maklumat UTHM

Training Materials

https://bit.ly/git_training_fsktm

Contents

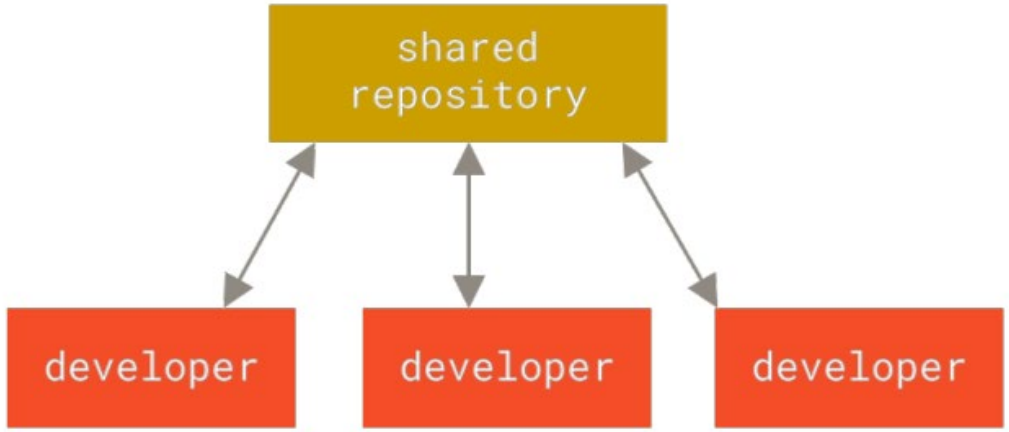
Why we need Source / Version control

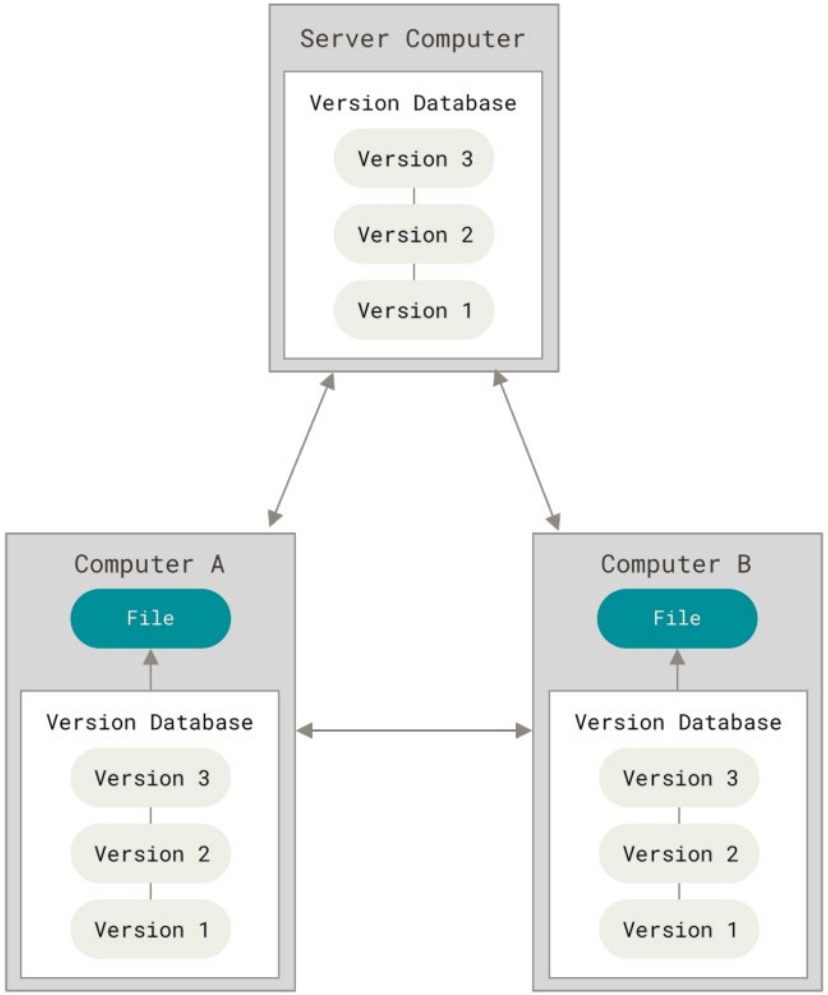
- Every software projects implemented in source code.
- Source code should be treated with care.
- We want them to be:
  - Safe
  - Retain a history of changes
  - Attribute credit (or blame!) to the authors
  - Allow us to collaborate with other developers

Source Control / Version Control

- Practice of tracking and managing changes to code
- Source Control Management (SCM) systems:
  - Allow tracking of code changes
  - See revision history
  - Revert to previous version of project when needed
  - Collaborate with all developers
  - Centralized or Distributed types
- Example of popular SCM tools:
  - *CVS*
  - *Microfocus / Borland Starteam*
  - *Apache Subversion (SVN)*
  - *Perforce Helix*
  - *Visual SourceSafe*
  - *Mercurial*
  - *Rational ClearCase*

Centralized vs Distributed





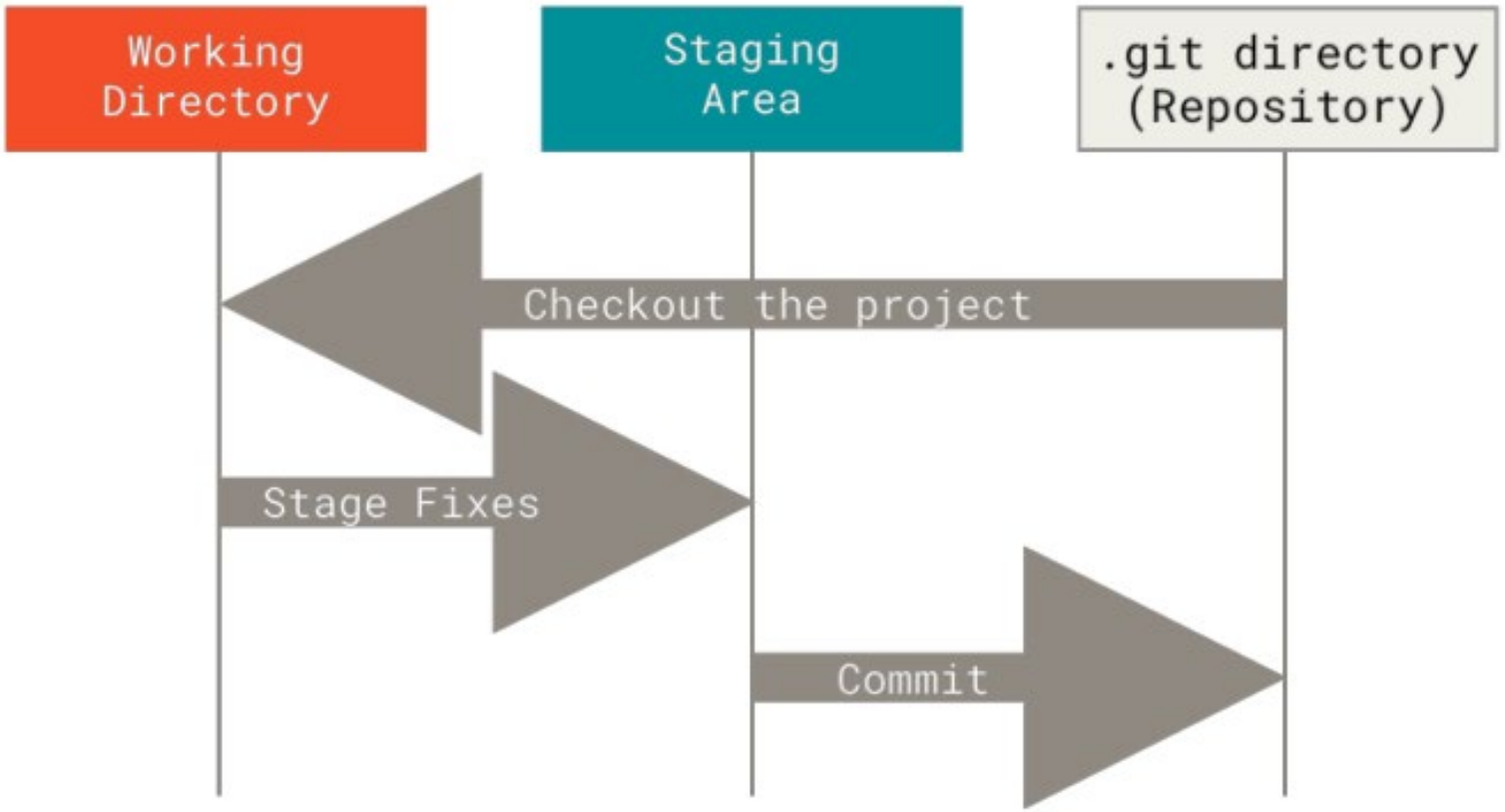
Git Introduction

- Git development goals:
  - Speed
  - Simple design
  - Strong support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects like the Linux kernel efficiently
- Created originally by **Linus Torvalds** in 2005 for Linux kernel development
- A free and open-source distributed version control systems
- Distributed means every Git directory on every computer is a full-fledged repository with complete history and full version tracking abilities

Git Benefits

- **Historical Change Tracking**
  - You can review a graph of how your commits have changed over time, see when and by whom changes were made, and revert to a previous commit if needed. This history makes it easier to identify and fix bugs.
- **Work as a Team**
  - You can easily share your code with teammates for review before you commit or merge back to the main working branch. Additionally, the branching and review capabilities enable simultaneous development. Multiple people can work on the same file and resolve differences later.
- **Improve Team Speed & Productivity**
  - Git makes it easy for your team to track changes to your code. Now you can focus on writing code instead of spending time tracking and merging different versions across your team. Additionally, Git performs computations and stores your main repository locally, making it quicker on most operations than a centralized SCM.
- **Availability and Redundancy**
  - Git is a distributed SCM, meaning there is no single, central place where everything is stored. In a distributed system, there are multiple backups in the event that you need one. This approach also means that you can work offline and commit your changes when you're ready.
- **Git is the Industry Standard**
  - Due to its popularity, Git is supported by many integrated development environments (IDE) and many popular developer tools

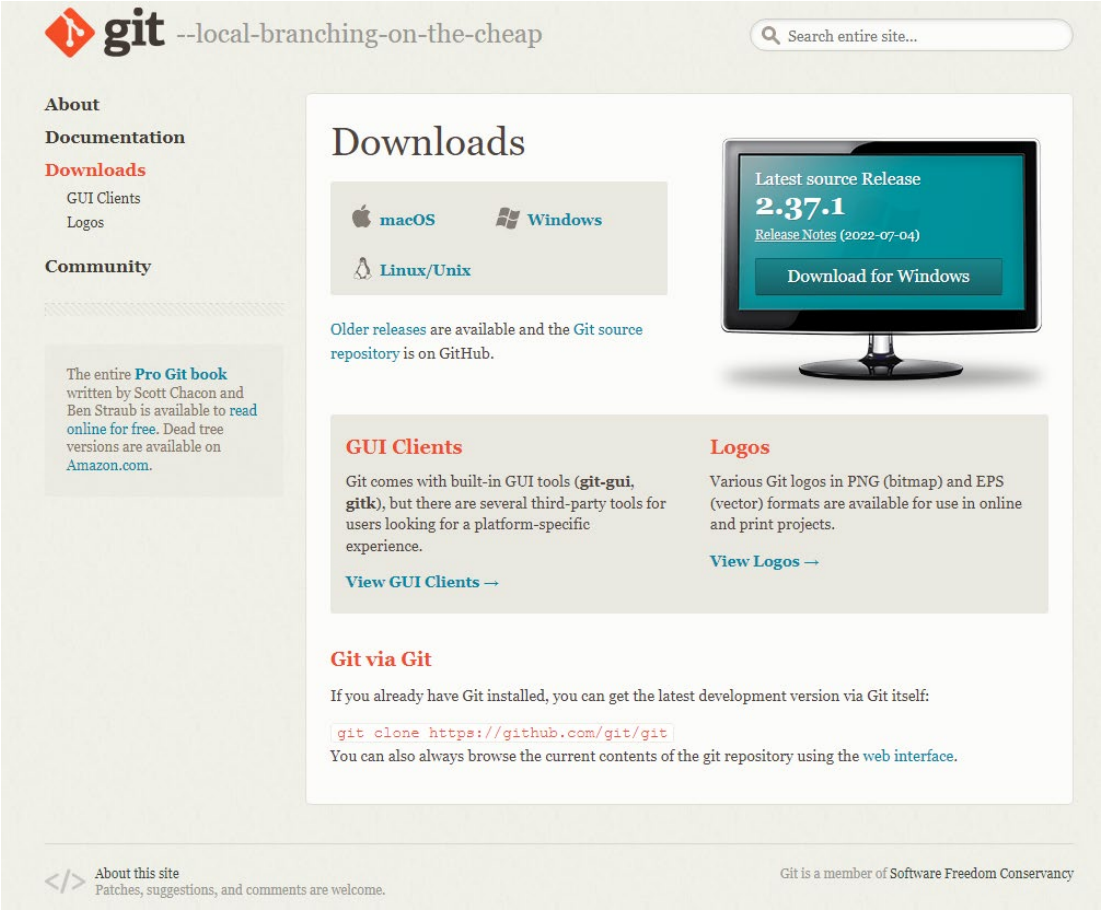
The Three States



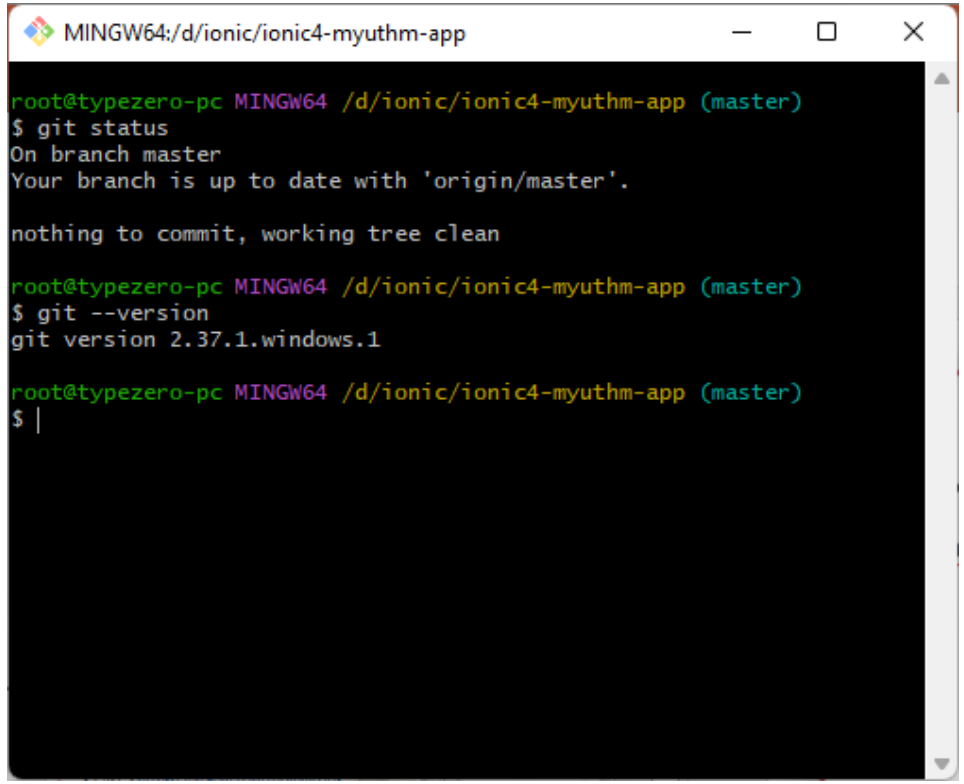
- **Modified**
  - You have changed the file but have not committed it to your database
- **Staged**
  - You have marked a modified file in its current version to go into your next commit snapshot
- **Committed**
  - The data is safely stored in your database

Download & Install Git

- URL : https://git-scm.com/downloads
- Choose version for your platform
  - Windows
    - 32bit or 64bit
  - Linux
  - Mac



Git Command Lines



- Command line is the only place that you can run **all** Git commands.
- Most of the GUIs implement a partial subset of Git functionality.
- If you know how to run the command line version, you probably can figure out how to run the GUI version.
- For Windows users, you can use Command Prompt or PowerShell.
- For Mac users, you can use Terminal.

Git Command Lines

- Setup & Config
  - `git config`
  - `git help`
- Getting & Creating Projects
  - `git init`
  - `git clone`
- Basic Snapshotting
  - `git add`
  - `git status`
  - `git diff`
  - `git difftool`
  - `git commit`
  - `git reset`
  - `git rm`
  - `git mv`
  - `git clean`
- Branching & Merging
  - `git branch`
  - `git checkout`
  - `git merge`
  - `git mergetool`
  - `git log`
  - `git stash`
  - `git tag`
- Sharing & Updating Project
  - `git fetch`
  - `git pull`
  - `git push`
  - `git remote`
  - `git archive`
  - `git submodule`
- Inspection & Comparison
  - `git show`
  - `git shortlog`
  - `git describe`
- Debugging
  - `git bisect`
  - `git blame`
  - `git grep`
- Patching
  - `git cherry-pick`
  - `git rebase`
  - `git revert`

Getting Help

```
C:\> git --version

C:\> git --help <verb>

C:\> git <verb> -h
```

Initialize from existing local project

```
C:\> cd C:\training\project1
C:\training\project1> git init
Initialized empty Git repository in
C:/training/project1/.git/
```

- This create a new subdirectory named `.git`
- It contains all necessary repository files

Initialize from a remote repository

```
C:\training\project1> git clone
https://git.uthm.edu.my/training.git
```

- This will clone a repository into a new directory

Configuration

```
C:\training\project1> git config --global user.name “Test User”
C:\training\project1> git config --global user.email test@uthm.edu.my
C:\training\project1> git config --list
C:\training\project1> git config --global alias.hist "log --
pretty=format:'%h %ad | %s%d [%an]' --graph --date=short"
```

- This command is used to list and customize your Git environment and save it as a configuration variable.
- Typically it will save the configuration variables inside:
  - ~/.gitconfig
  - .git/con
  - <path>/e

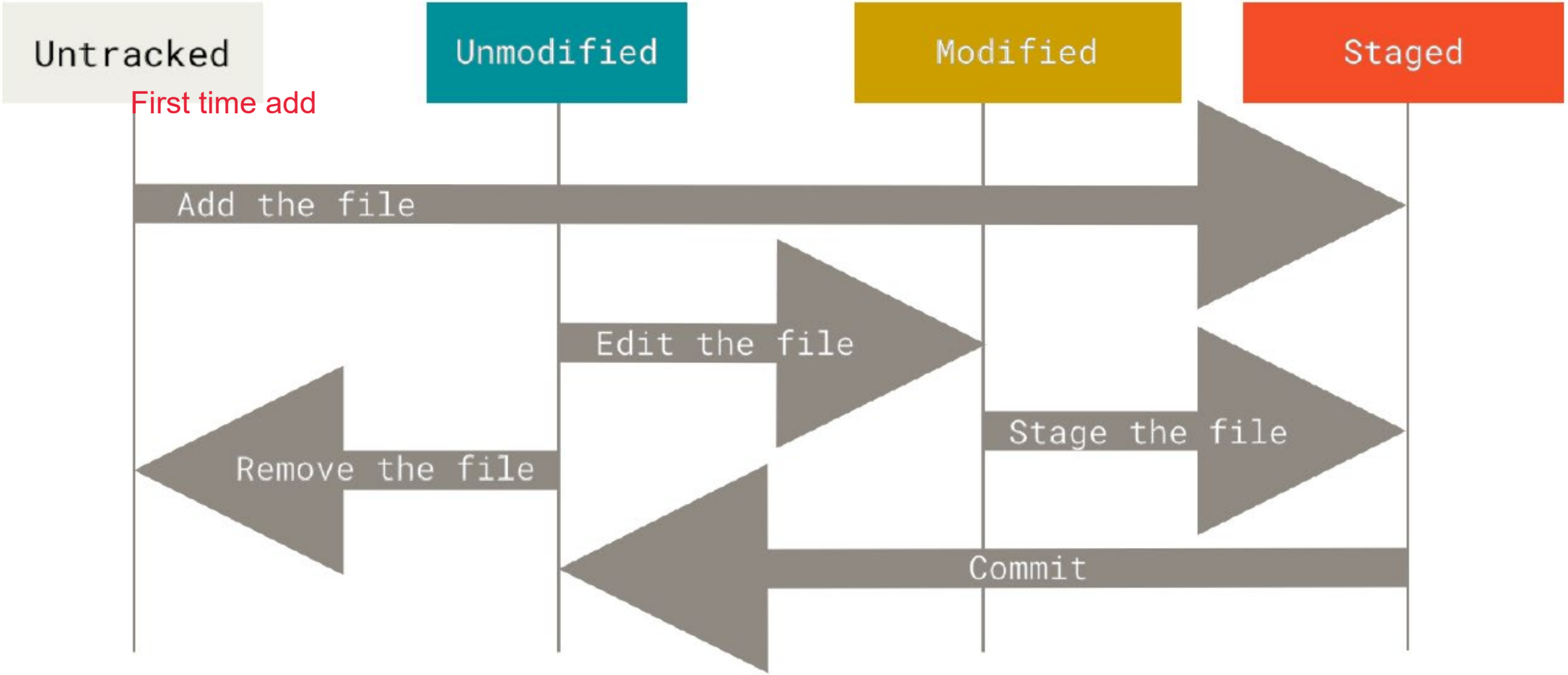
```
C:\training_GitCourse\project1>git config --global alias.hist "log --pretty=format:'%h %ad | %s%d [%an]' --graph
--date=short"

C:\training_GitCourse\project1>git hist
* e541c7b 2022-08-08 | VersionSyuk_0.03 (HEAD -> master) [Syukri Yazed]
* 4b1c331 2022-08-08 | VersionSyuk_0.02 [Syukri Yazed]
* d42f6d4 2022-08-08 | VersionSyuk_0.01 [Syukri Yazed]
```

Add file contents to repository index

```
C:\training\project1> git add index.php
C:\training\project1> git add *.php
```

- This command updates the index using the current content found in the working tree, to prepare the content staged for the next commit.
- The "index" holds a snapshot of the content of the working tree, and it is this snapshot that is taken as the contents of the next commit.
- Can be performed multiple times before a commit.

The Lifecycle of File Status



Ignoring files

```
C:\training\project1> type .gitignore
# ignore all .a files
*.a
# but do track lib.a, even though you're ignoring .a files above
!lib.a
# only ignore the TODO file in the current directory, not subdir/TODO
/TODO
# ignore all files in any directory named build
build/
# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt
# ignore all .pdf files in the doc/ directory and any of its subdirectories
doc/**/*.pdf
```

- Create a file listing patterns and name it as `.gitignore`

Show the working tree status

```
C:\training\project1> git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean
```

- This command will show the different states of files in your working directory and staging area.

```
C:\training_GitCourse\project1>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   gallery.html
        new file:   index.html
        new file:   links.html
        new file:   resume.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        README.txt
```

See what is modified but unstaged

```
C:\training\project1> git diff
diff --git a/gallery.html b/gallery.html
index e69de29..30d74d2 100644
--- a/gallery.html
+++ b/gallery.html
@@ -0,0 +1 @@
+test
\ No newline at end of file
```

- This command will show the changes between commits, commits and working tree, etc

Record changes to the repository

```
C:\training\project1> git commit –m "First commit"
```

- This command takes all the file contents that have been staged with `git add` and records a permanent snapshot in the database
- It will then moves the branch pointer on the current branch up to it.

```
C:\training_GitCourse\project1>git commit -m "VersionSyuk_0.01"
[master (root-commit) d42f6d4] VersionSyuk_0.01
 4 files changed, 3 insertions(+)
 create mode 100644 gallery.html
 create mode 100644 index.html
 create mode 100644 links.html
 create mode 100644 resume.html
```

Commit Checksums

- Git generates a unique SHA-1 hash (40 string of hex digits) for every commits.
- Git refers commits by this ID rather than a version number.
- Often we only see the first 7 characters:

```
C:\training_GitCourse\project1>git log
commit e541c7be26dcfe2ff9c1ee40db05ecfeaafff831 (HEAD -> master)
Author: Syukri Yazed <hi200006@siswa.uthm.edu.my>
Date:   Mon Aug 8 10:47:56 2022 +0800

    VersionSyuk_0.03

commit 4b1c331c20c30cd48130a4f53171d2d8ef557bd2
Author: Syukri Yazed <hi200006@siswa.uthm.edu.my>
Date:   Mon Aug 8 10:43:50 2022 +0800

    VersionSyuk_0.02

commit d42f6d4ca7db6ce69de20f56974c5be7267f6c93
Author: Syukri Yazed <hi200006@siswa.uthm.edu.my>
Date:   Mon Aug 8 10:37:32 2022 +0800

    VersionSyuk_0.01
```

```
st line of readme
 to readme
mmit
```

```
C:\training_GitCourse\project1>git log
commit d42f6d4ca7db6ce69de20f56974c5be7267f6c93 (HEAD -> master)
Author: Syukri Yazed <hi200006@siswa.uthm.edu.my>
Date:   Mon Aug 8 10:37:32 2022 +0800

    VersionSyuk_0.01
```

Viewing commit history

```
C:\training\project1> git log
C:\training\project1> git log --pretty=oneline
C:\training\project1> git log --pretty=oneline --max-count=2
C:\training\project1> git log --pretty=oneline --since='5 minutes ago'
C:\training\project1> git log --pretty=oneline --author=<your name>
C:\training\project1> git log --pretty=oneline --all
C:\training\project1> git log --pretty=format:"%h %ad | %s%d [%an]" --graph --date=short
```

- This command will show commits log

```
C:\training_GitCourse\project1>git log
commit e541c7be26dcfe2ff9c1ee40db05ecfeaafff831 (HEAD -> master)
Author: Syukri Yazed <hi200006@siswa.uthm.edu.my>
Date:   Mon Aug 8 10:47:56 2022 +0800

    VersionSyuk_0.03

commit 4b1c331c20c30cd48130a4f53171d2d8ef557bd2
Author: Syukri Yazed <hi200006@siswa.uthm.edu.my>
Date:   Mon Aug 8 10:43:50 2022 +0800

    VersionSyuk_0.02

commit d42f6d4ca7db6ce69de20f56974c5be7267f6c93
Author: Syukri Yazed <hi200006@siswa.uthm.edu.my>
Date:   Mon Aug 8 10:37:32 2022 +0800

    VersionSyuk_0.01
```

Checkout older version

```
C:\training\project1> git checkout <hash>
```

- Refer to previous version hash ID
- `<hash>` can be found from `git log` command

Tagging files

```
C:\training\project1> git tag
```

- This command is used to create, list, delete or verify a tag (a specific points in a repository history)
- Usually, developers use this functionality to mark release points (`v1.0, v2.0` and so on)

Discard changes of a modified file

```
C:\training\project1> git checkout test.php
C:\training\project1> git restore test.php
```

- This command is **dangerous**.
- Any local changes you made to that file are gone.
- Git just replaced that file with the last staged or committed version.
- Don't ever use this command unless you absolutely know that you don't want those unsaved local changes.

Removing files from repository

```
C:\training\project1> del index.php
C:\training\project1> git rm index.php
```

- This command is used to remove files from the staging area and working directory.
- Similar to `git add` in the that is stages a removal of a file for the next commit.

Branching

```
C:\training\project1> git branch --list
C:\training\project1> git branch <branchname>
C:\training\project1> git branch -d <branchname>
C:\training\project1> git checkout -b <branchname>
```

- This command will list, create or delete branches

Join development changes

```
C:\training\project1> git merge
```

- This command is used to merge one or more branches into the branch you have checkout.

Fetching from and integrate with another repository or a local branch

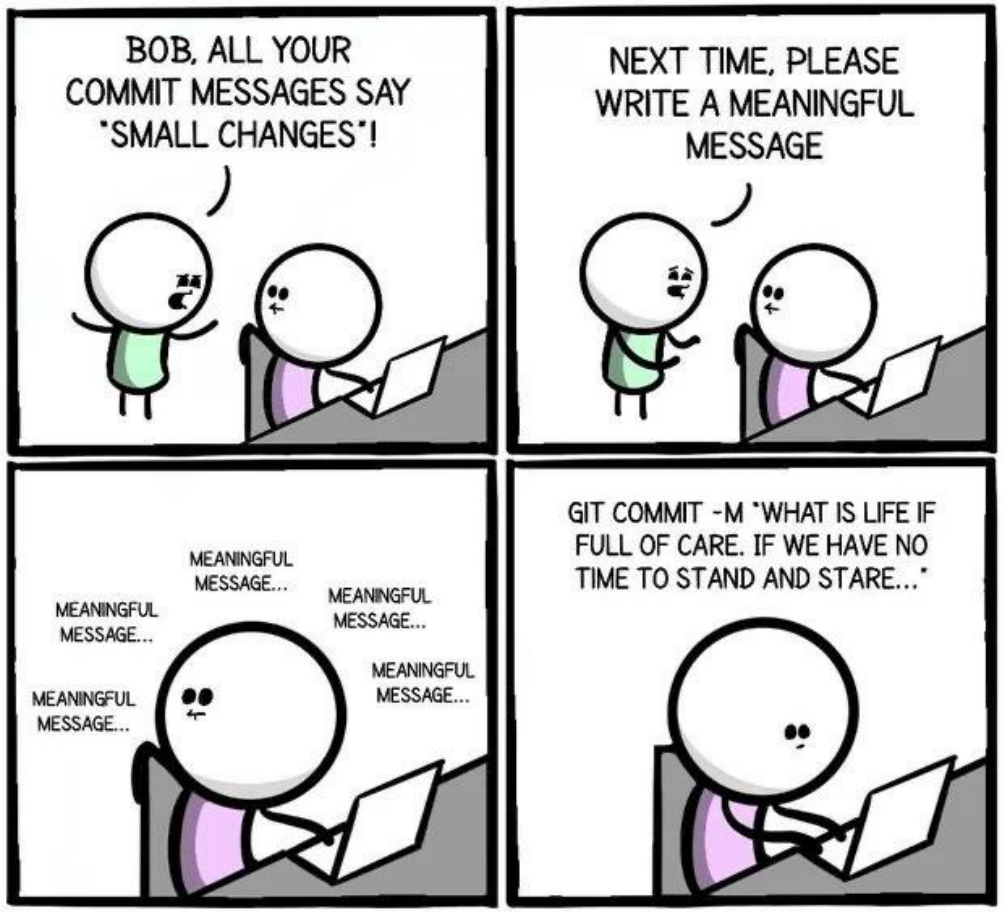
```
C:\training\project1> git pull origin master
```

- Incorporates changes from a remote repository into the current branch.
- If the current branch is behind the remote, then by default it will fast-forward the current branch to match the remote.

Update remote repository

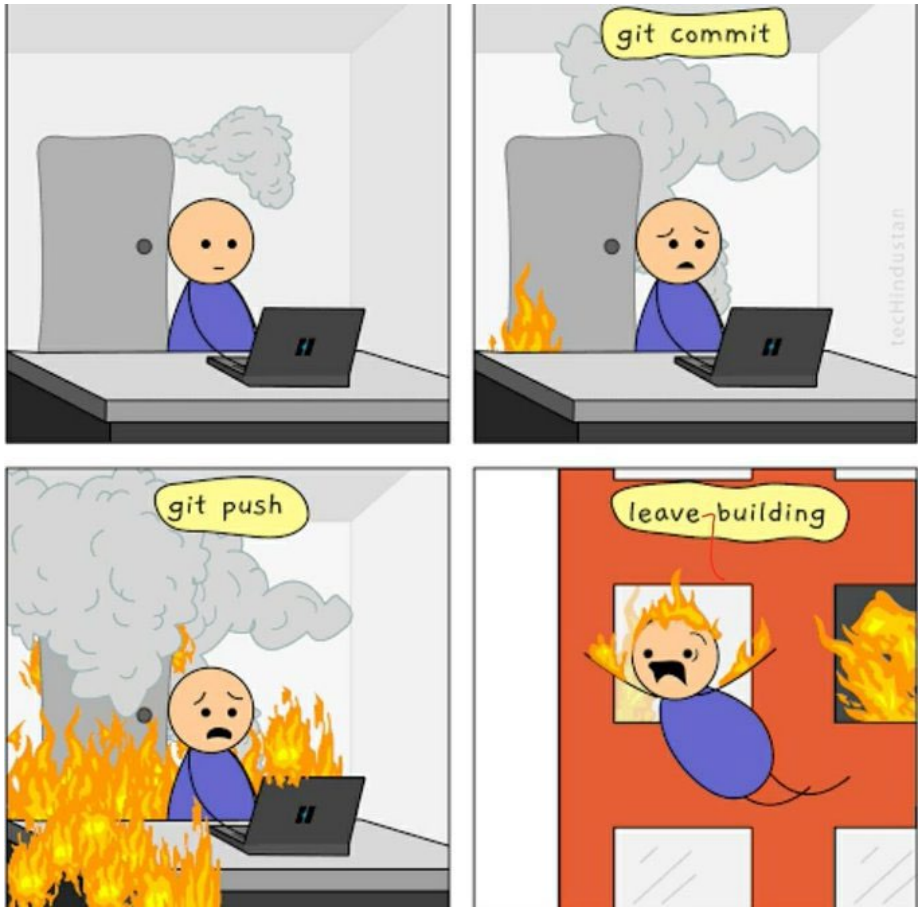
```
C:\training\project1> git push origin master
```

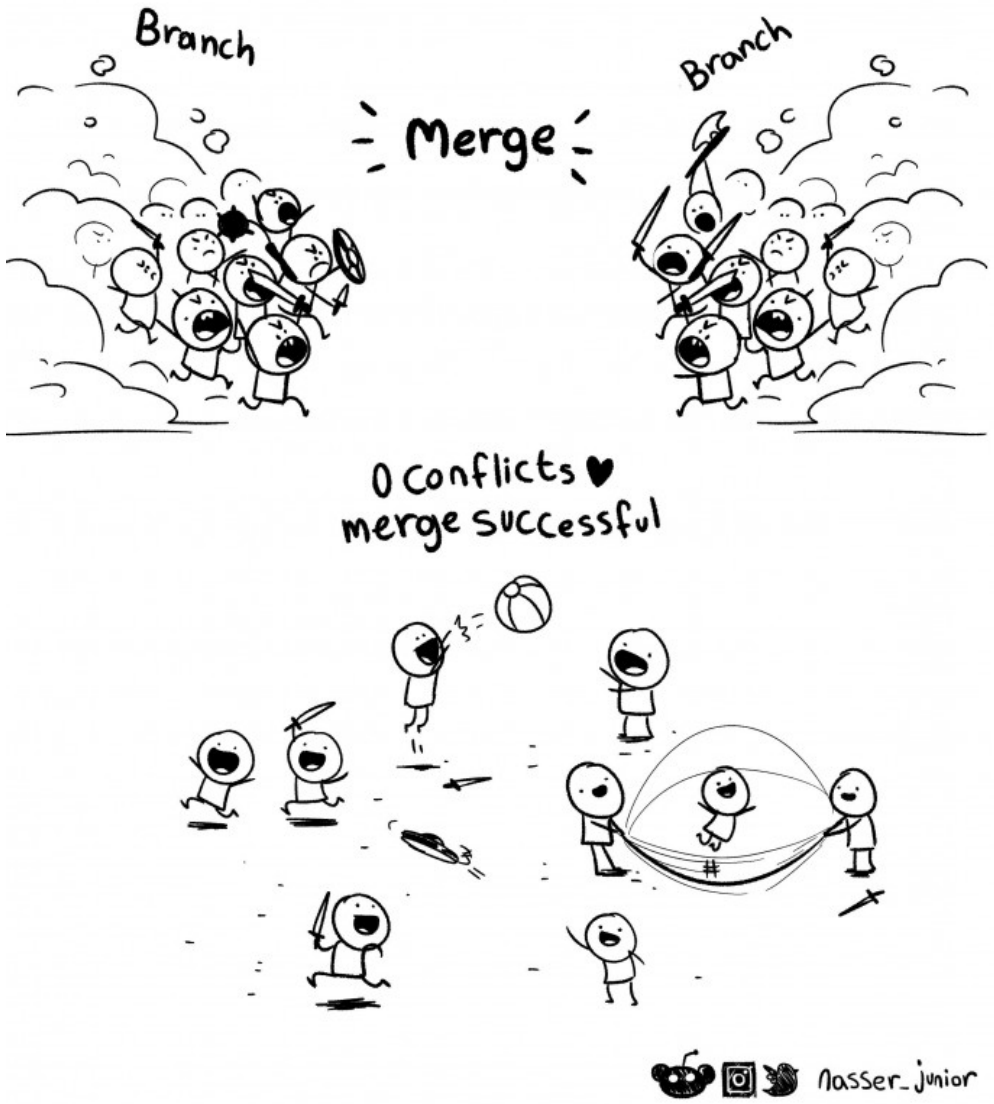
- This command is used to communicate with another repository, calculate what your local database the remote one does note and then pushes the difference into the other repository.


BOB, ALL YOUR COMMIT MESSAGES SAY "SMALL CHANGES"!
NEXT TIME, PLEASE WRITE A MEANINGFUL MESSAGE
MEANINGFUL MESSAGE...
MEANINGFUL MESSAGE...
MEANINGFUL MESSAGE...
MEANINGFUL MESSAGE...
MEANINGFUL MESSAGE...
GIT COMMIT -M "WHAT IS LIFE IF FULL OF CARE. IF WE HAVE NO TIME TO STAND AND STARE..."
@_workchronicles
workchronicles.com
follow on Twitter & Instagram for more (and don't forget the underscore)


BLAME
WHO WROTE THIS CRAP?
WAS IT ME?
NOT ME
MONKEYUSER.COM


In case of fire
1. git commit
2. git push
3. leave building


git commit
git push
leave building



Here's a joke for you...

Whenever I see a door that says "push", I always pull first, to avoid conflicts.

CREATE

Clone an existing repository

```
$ git clone ssh://user@domain.com/repo.git
```

Create a new local repository

```
$ git init
```

LOCAL CHANGES

Changed files in your working directory

```
$ git status
```

Changes to tracked files

```
$ git diff
```

Add all current changes to the next commit

```
$ git add .
```

Add some changes in <file> to the next commit

```
$ git add -p <file>
```

Commit all local changes in tracked files

```
$ git commit -a
```

Commit previously staged changes

```
$ git commit
```

Change the last commit

Don't amend published commits!

```
$ git commit --amend
```

COMMIT HISTORY

Show all commits, starting with newest

```
$ git log
```

Show changes over time for a specific file

```
$ git log -p <file>
```

Who changed what and when in <file>

```
$ git blame <file>
```

BRANCHES & TAGS

List all existing branches

```
$ git branch -av
```

Switch HEAD branch

```
$ git checkout <branch>
```

Create a new branch based on your current HEAD)

```
$ git branch <new-branch>
```

Create a new tracking branch based on a remote branch

```
$ git checkout --track <remote/branch>
```

Delete a local branch

```
$ git branch -d <branch>
```

Mark the current commit with a tag

```
$ git tag <tag-name>
```

UPDATE & PUBLISH

List all currently configured remotes

```
$ git remote -v
```

Show information about a remote

```
$ git remote show <remote>
```

Add new remote repository, named <remote>

```
$ git remote add <shortname> <url>
```

Download all changes from <remote>, but don't integrate into HEAD

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

Download changes and directly merge/integrate into HEAD

```
$ git pull <remote> <branch>
```

Publish local changes on a remote

```
$ git push <remote> <branch>
```

Delete a branch on the remote

```
$ git branch -dr <remote/branch>
```

Publish your tags

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

MERGE & REBASE

Merge <branch> into your current HEAD

```
$ git merge <branch>
```

Rebase your current HEAD onto <branch>

Don't rebase published commits!

```
$ git rebase <branch>
```

Abort a rebase

```
$ git rebase --abort
```

Continue a rebase after resolving conflicts

```
$ git rebase --continue
```

Use your configured merge tool to solve conflicts

```
$ git mergetool
```

Use your editor to manually solve conflicts and (after resolving) mark file as resolved

```
$ git add <resolved-file>
$ git rm <resolved-file>
```

UNDO

Discard all local changes in your working directory

```
$ git reset --hard HEAD
```

Discard local changes in a specific file

```
$ git checkout HEAD <file>
```

Revert a commit (by producing a new commit with contrary changes)

```
$ git revert <commit>
```

Reset your HEAD pointer to a previous commit ...and discard all changes since then

```
$ git reset --hard <commit>
```

...and preserve all changes as unstaged changes

```
$ git reset <commit>
```

...and preserve uncommitted local changes

```
$ git reset --keep <commit>
```

Git Cheat Sheet

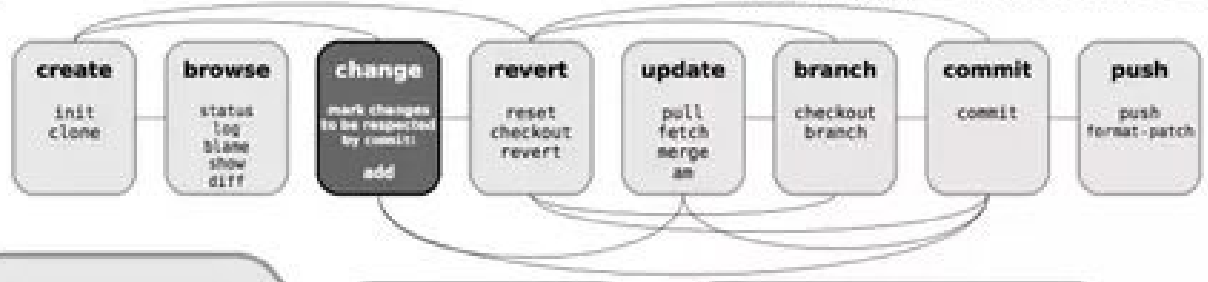
by Jan Krüger <jk@jk.gs>, http://jan-krueger.net/git/
Based on work by Zack Rusin

Basics

Use git help [command] if you're stuck.

master	default devel branch
origin	default upstream branch
HEAD	current branch
HEAD^	parent of HEAD
HEAD~4	great-great grandparent of HEAD
foo..bar	from branch foo to branch bar

(left to right) Command Flow



Create

From existing files

```
git init
git add .
```

From existing repository

```
git clone ~/old ~/new
git clone git://...
git clone ssh://...
```

View

```
git status
git diff [oldid newid]
git log [-p] [file|dir]
git blame file
git show id (meta data + diff)
git show id:file
git branch (shows list, * = current)
git tag -l (shows list)
```

Revert

In Git, revert usually describes a new commit that undoes previous commits.

```
git reset --hard (NO UNDO)
    (reset to last commit)
git revert branch
git commit -a --amend
    (replaces prev. commit)
git checkout id file
```

Publish

In Git, commit only respects changes that have been marked explicitly with add.

```
git commit [-a]
    (-a: add changed files automatically)
git format-patch origin
    (create set of diffs)
git push remote
    (push to origin or remote)
git tag foo
    (mark current version)
```

Update

```
git fetch (from def. upstream)
git fetch remote
git pull (= fetch & merge)
git am -3 patch.mbox
git apply patch.diff
```

Branch

```
git checkout branch
    (switch working dir to branch)
git merge branch
    (merge into current)
git branch branch
    (branch current)
git checkout -b new other
    (branch new from other and switch to it)
```

Useful Tools

```
git archive
    Create release tarball
git bisect
    Binary search for defects
git cherry-pick
    Take single commit from elsewhere
git fsck
    Check tree
git gc
    Compress metadata (performance)
git rebase
    Forward-port local changes to remote branch
git remote add URL
    Register a new remote repository for this tree
git stash
    Temporarily set aside changes
git tag
    (there's more to it)
gitk
    Tk GUI for Git
```

Conflicts

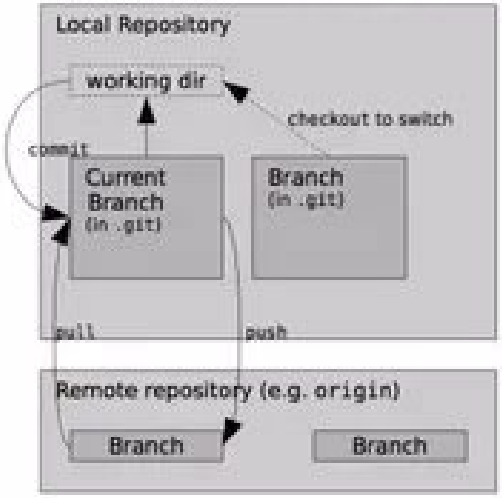
Use add to mark files as resolved.

```
git diff [--base]
git diff --ours
git diff --theirs
git log --merge
gitk --merge
```

Tracking Files

```
git add files
git mv old new
git rm files
git rm --cached files
    (stop tracking but keep files in working dir)
```

Structure Overview



git cheat sheet

Initialization

$ git init <directory>
Creates new repo in specified directory

$ git clone <url>
Copies repo from specied url

$ git config user.name <user_name>
Sets username for commits in current repo
Use --global to apply it globally

$ git config user.email <user_email>
Sets email for commit in current repo
Use --global to apply it globally

$ git config color.ui auto
Enables helpful colorisation of command line
output

$ git config --global --edit
Opens the global configuration file in text editor for manual editing

$ git remote add origin <link>
Connects your local repo to the remote one

$ vi .gitignore
Opens .gitignore file. This file is used for list of files that have to be excluded. Ensure that this file is in root of local repo. You can change **vi** into your favourite text editor

Commits

$ git add <path>
Adds path into staging. Path can be file or directory

$ git restore --staged <path>
Removes path from staging back to unstaged area

$ git rm -r <path>
Removes path and adds that change into staging

$ git commit -m <message>
Commits the stage with specified message

$ git commit --amend -m <message>
Repairs last commit with specified new message

$ git commit --amend --no-edit
Repairs last commit without editing commit message

$ git status
Lists which files are staged, unstaged, or untracked

$ git push
Uploads all commits to remote branch

$ git pull -r
Updates local branch with all new commits from remote branch with rebasing, avoiding the conflict with changes from remote

Change Review

$ git log
Lists version history for the current branch

$ git diff <commit1> <commit2>
Shows difference between two commits. It is also applied to comparing two branches

$ git diff <commit1> <commit2> --name-only
Same with above, but only show the file names only

$ git stash
Save current changes into stash stack
Usually used when current changes don't want to be commited

$ git stash pop
Applies last changes stored in stash stack onto current working HEAD

$ git stash list
Shows stash stack

$ git revert <commit>
Creates new commit that undoes all of the changes in <commit>

$ git reset <commit>
Undoes the commits after <commit>, keep the changes locally. Add --hard to discard the changes

Branch & Rebase

$ git checkout <branch>
Switches to the specified branch

$ git checkout -
Switches to the previous visited branch

$ git checkout -b <name>
Creates a new branch with specified name and switch in that branch

$ git checkout <path>
Restores changes of <path> back into latest revision

$ git branch
Lists all branchs

$ git branch -m <old> <new>
Renames branch from <old> to <new>

$ git branch -d <branch>
Deletes the specified branch

$ git rebase -i <base>
Interactively rebases the current branch onto base. It can be branch, commit, or relative reference to HEAD

$ git push -f
Uploads all commits to remote branch with force. Usually used when there are conflicts when rebasing. Do not try this unless you know what you are doing

Advanced

$ git checkout -R <old_branch> <new_branch>
$ git push origin :<old_branch> <new_branch>
Rename branch in local and remote correspondently

$ git tag <tag_name>
$ git push origin --tags
Create tag and push all created tags

$ git tag -d <tag_name>
$ git push origin --delete <tag_name>
Delete tag and push deleted tags

$ git remote set-url origin <url>
Changes remote url. Usually used after repository migration

$ git cherry-pick <commit>
Creates new commit by applying changes in <commit> into current working HEAD

$ git gc --prune=now --aggressive
Cleanups and optimizes all files in local repository

$ git bisect start
$ git bisect good
$ git bisect bad
Find the commit that contains bug with binary search

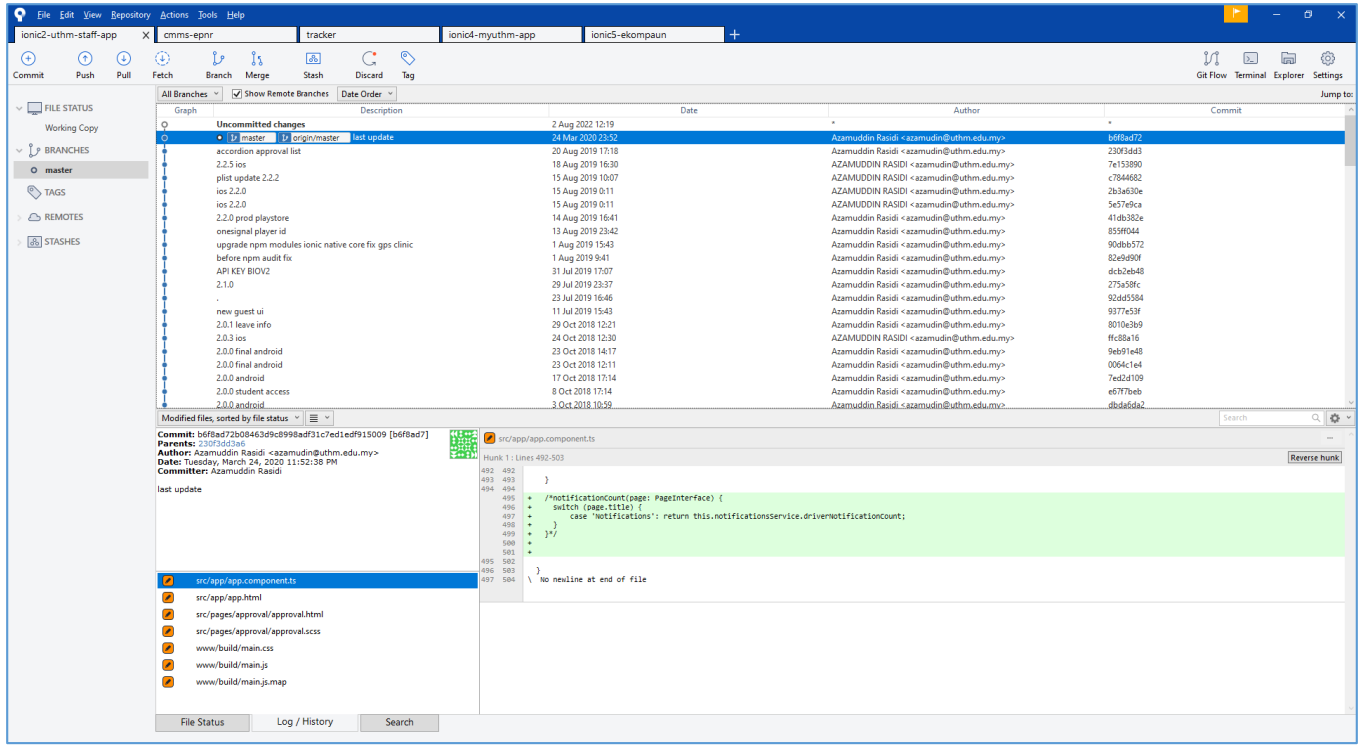
$ git blame -- <file>
Shows revision in <file> line by line

Git GUI Clients

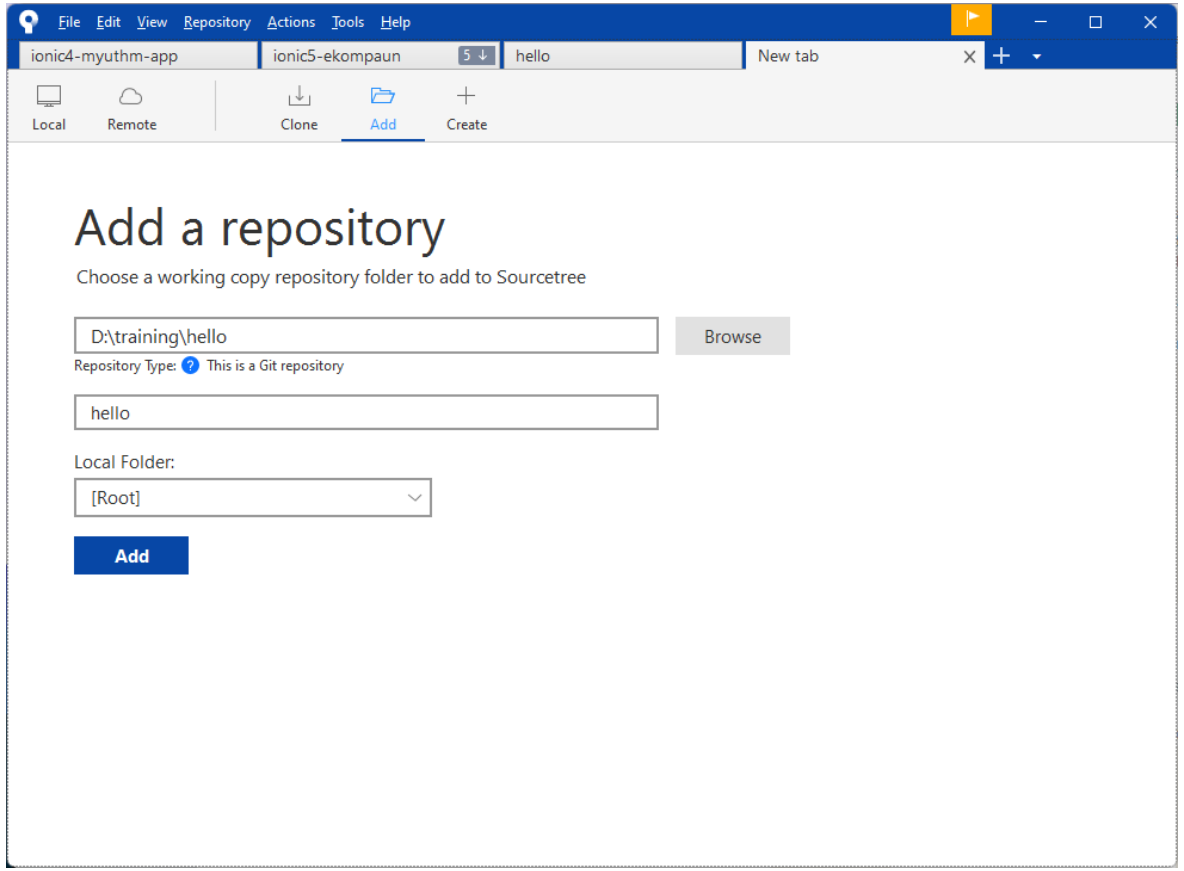
- List of GUI Clients :

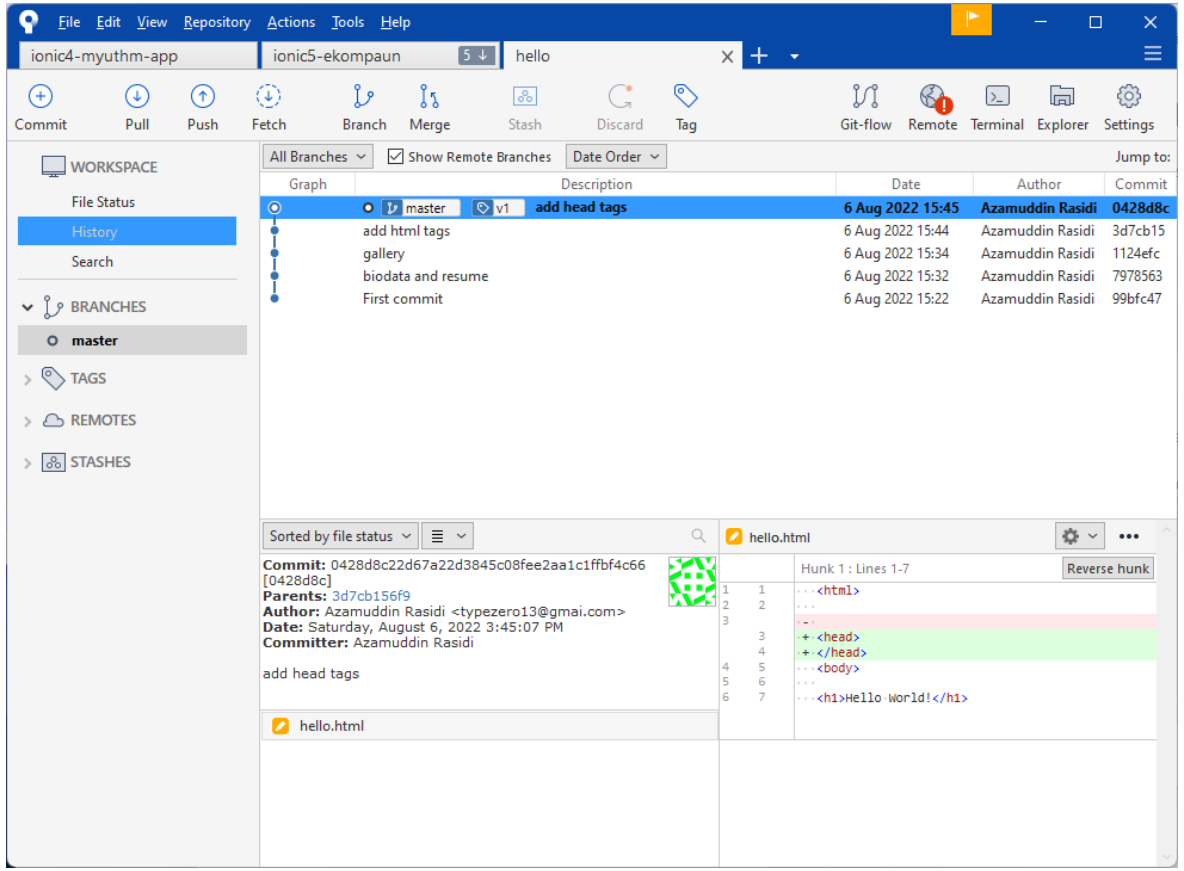
https://git-scm.com/downloads/guis

- For the purpose of today's training, we will use SourceTree
  - Download URL : https://www.sourcetreeapp.com
  - Windows and Mac versions are available.



SourceTree – Add a repository

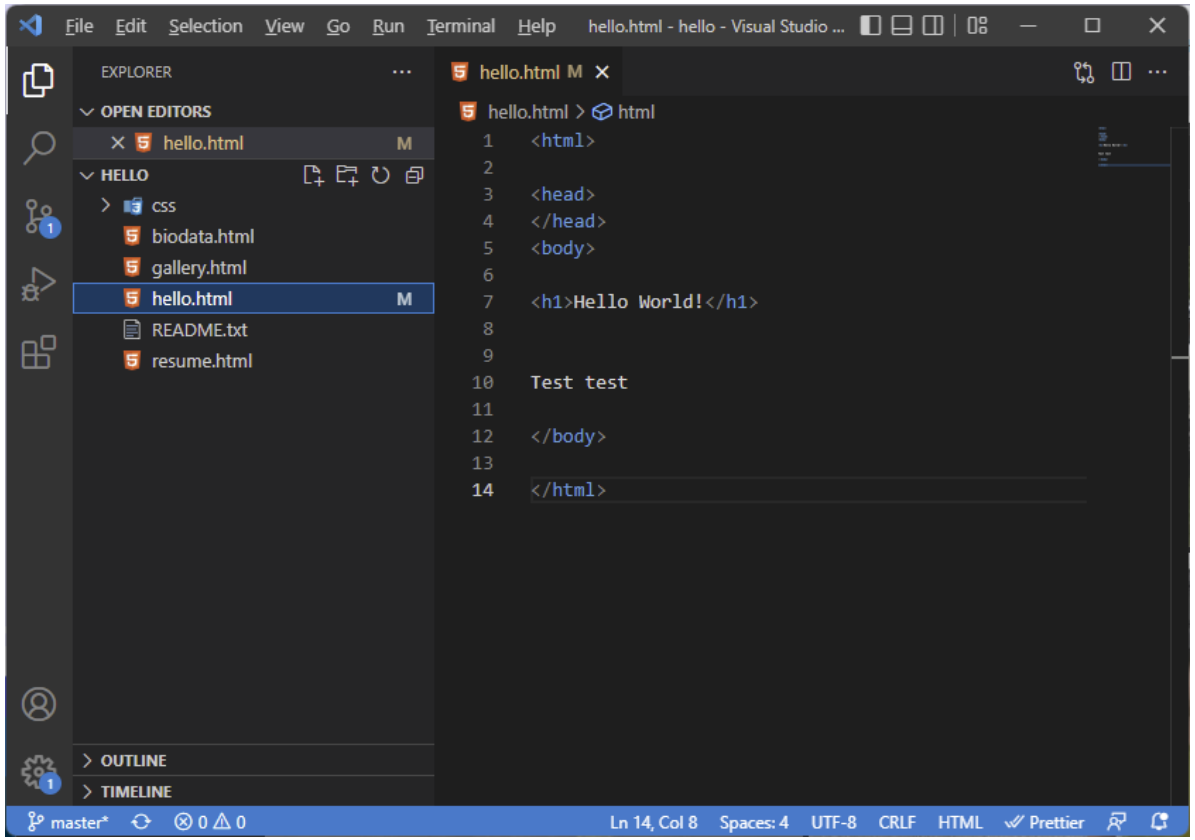


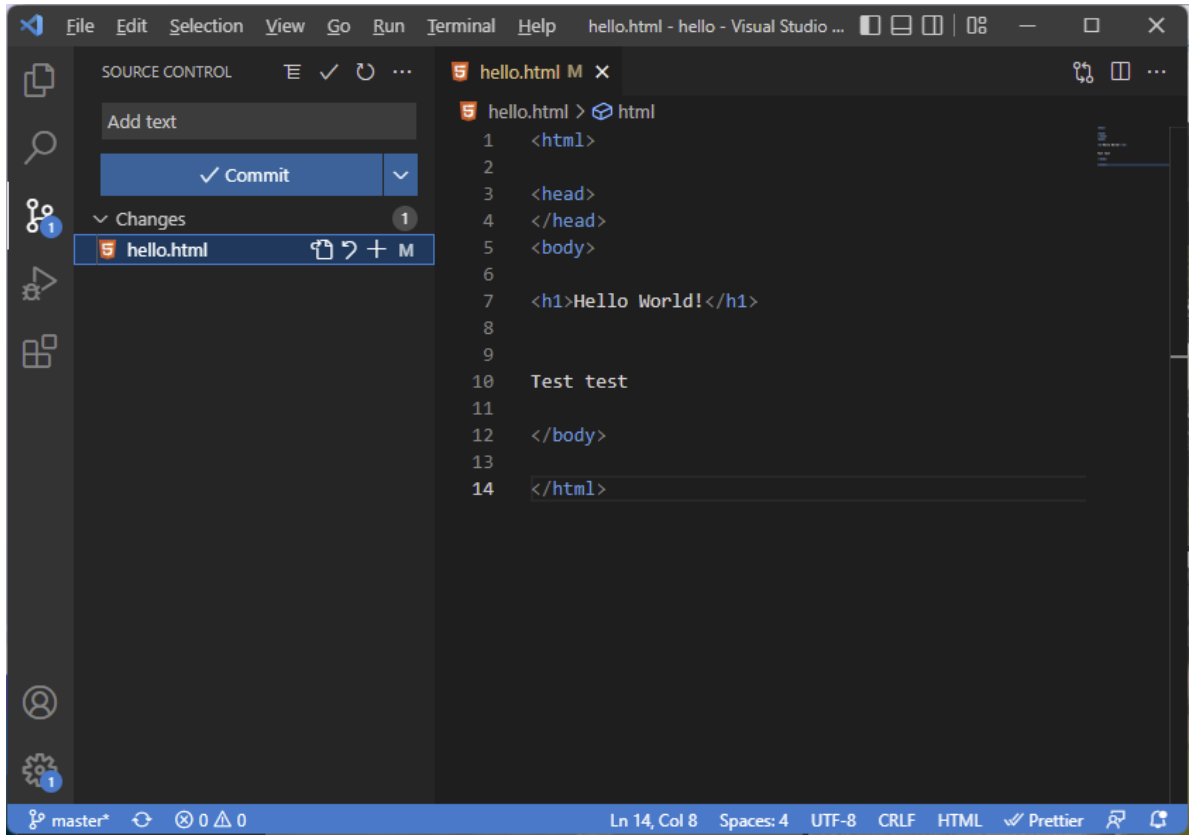


Git IDE Integration

- Most of the popular IDE used by developers now supports Git
- Among of the functionalities of Git that can be used inside the IDE
  - Create or clone a repository
  - Open and browse history of a repository
  - Create and checkout branches and tags
  - Stash, stage, and commit changes
  - Fetch, pull, push, or sync commits
  - Merge and rebase branches
  - Resolve merge conflicts
  - View diffs
- Some of the IDE that integrates with Git
  - Visual Studio / Visual Studio Code
  - IntelliJ
  - PhpStorm
  - Sublime Text
  - PyCharm
  - Eclipse
  - etc...

Git IDE Integration





Remote Git Repository

- A Remote in Git is a common repository that developers/team members use to exchange their changes
- This can be a your own Git server that were hosted on the internet or a code repository service such as GitHub
- Some of the examples of code repository service
  - Github – https://www.github.com
  - BitBucket – https://www.bitbucket.org
  - GitLab – https://www.gitlab.com

Github

- An internet hosting service for software development and version control using Git.
- It provides :
  - Distributed version control
  - Access control
  - Bug tracking
  - Software feature request
  - Task management
  - Continuous integration (CI/CD)
  - Wiki
- Currently the largest source code hosting in the world.

Github

Search or jump to... / Pull requests Issues Marketplace Explore

Choose the plan that's right for you.

How often do you want to pay?

Monthly | Yearly Get 1 month free

Free

The basics for individuals and organizations

$0 per year forever

Create a free organization

- Unlimited public/private repositories
- Automatic security and version updates
- 2,000 CI/CD minutes/month
  Free for public repositories
- 500MB of Packages storage
  Free for public repositories
- New issues & projects (in limited beta)

MOST POPULAR

Team

Advanced collaboration for individuals and organizations

$48 $44 per user/year for the first 12 months*

Continue with Team

- Everything included in Free, plus...
- Access to GitHub Codespaces
- Protected branches
- Multiple reviewers in pull requests
- Draft pull requests
- Code owners

Enterprise

Security, compliance, and flexible deployment

$252 $231 per user/year for the first 12 months*

Start a free trial | Contact Sales

- Everything included in Team, plus...
- Enterprise Managed Users
- User provisioning through SCIM
- Enterprise Account to centrally manage multiple organizations
- Environment protection rules and secrets

Github - Create new repository

Create a new repository

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

Owner * typezero13 / Repository name * test

Great repository names are short and memorable. Need inspiration? How about animated-train?

Description (optional)

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

- Add a README file
  This is where you can write a long description for your project. Learn more.

Add .gitignore
Choose which files not to track from a list of templates. Learn more.
.gitignore template: None

Choose a license
A license tells others what they can and can't do with your code. Learn more.
License: None

You are creating a public repository in your personal account.

Create repository

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

Set up in Desktop or HTTPS SSH https://github.com/typezero13/training.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 "# training" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/typezero13/training.git
git push -u origin main
```

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

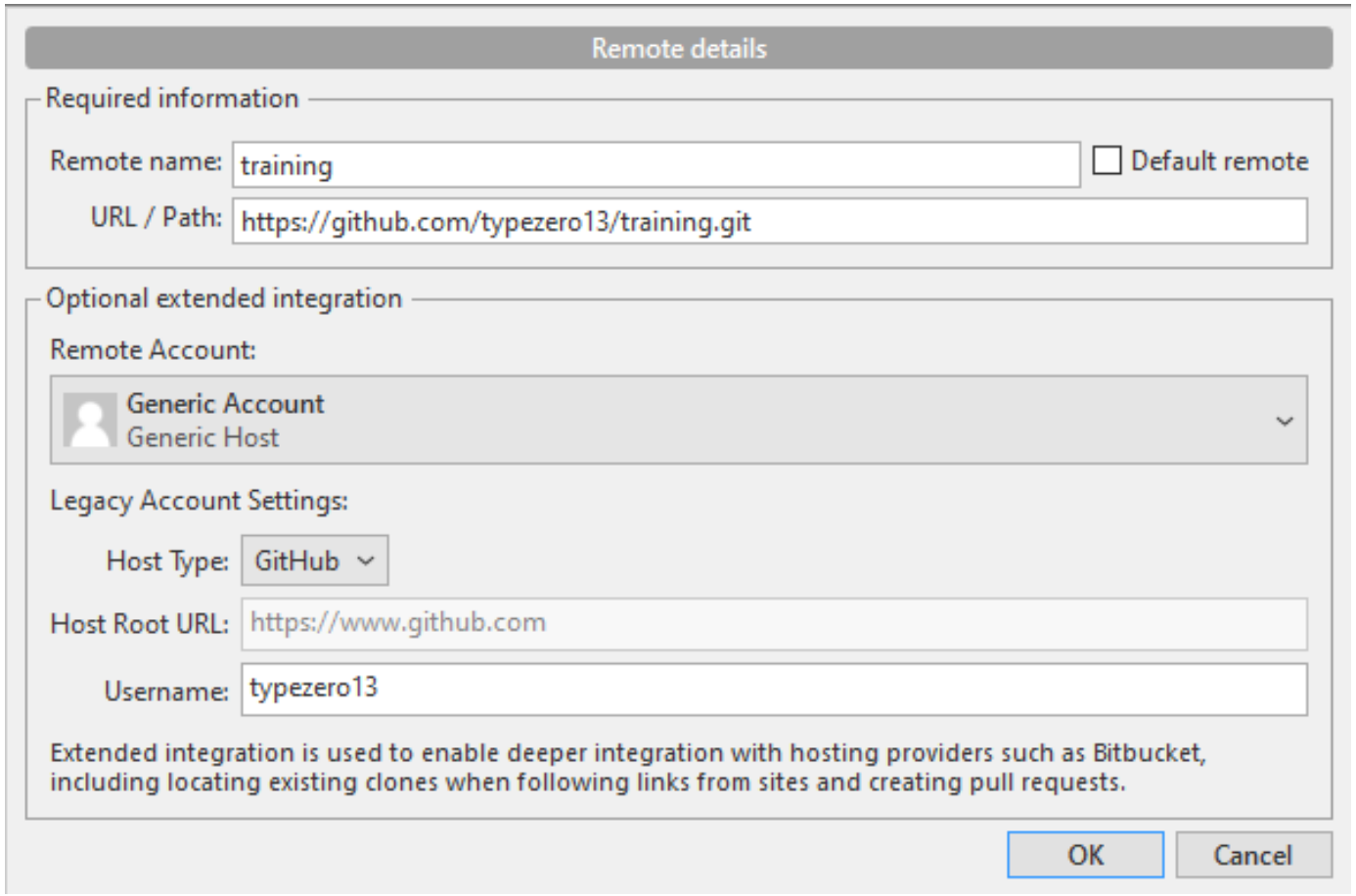
```
git remote add origin https://github.com/typezero13/training.git
git branch -M main
git push -u origin main
```

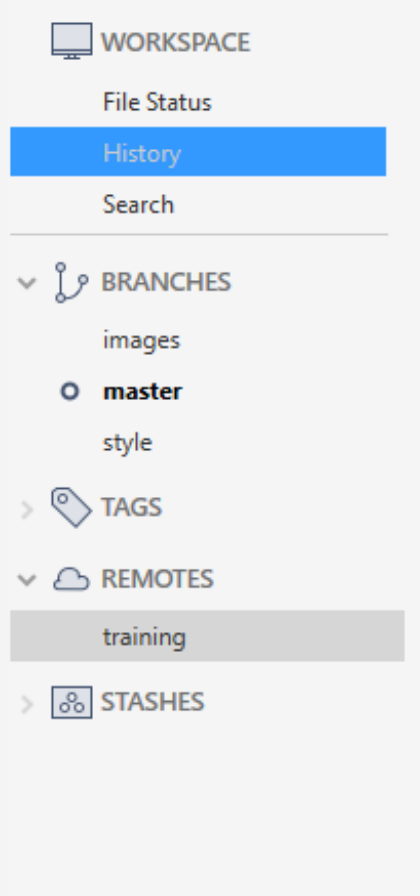
...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

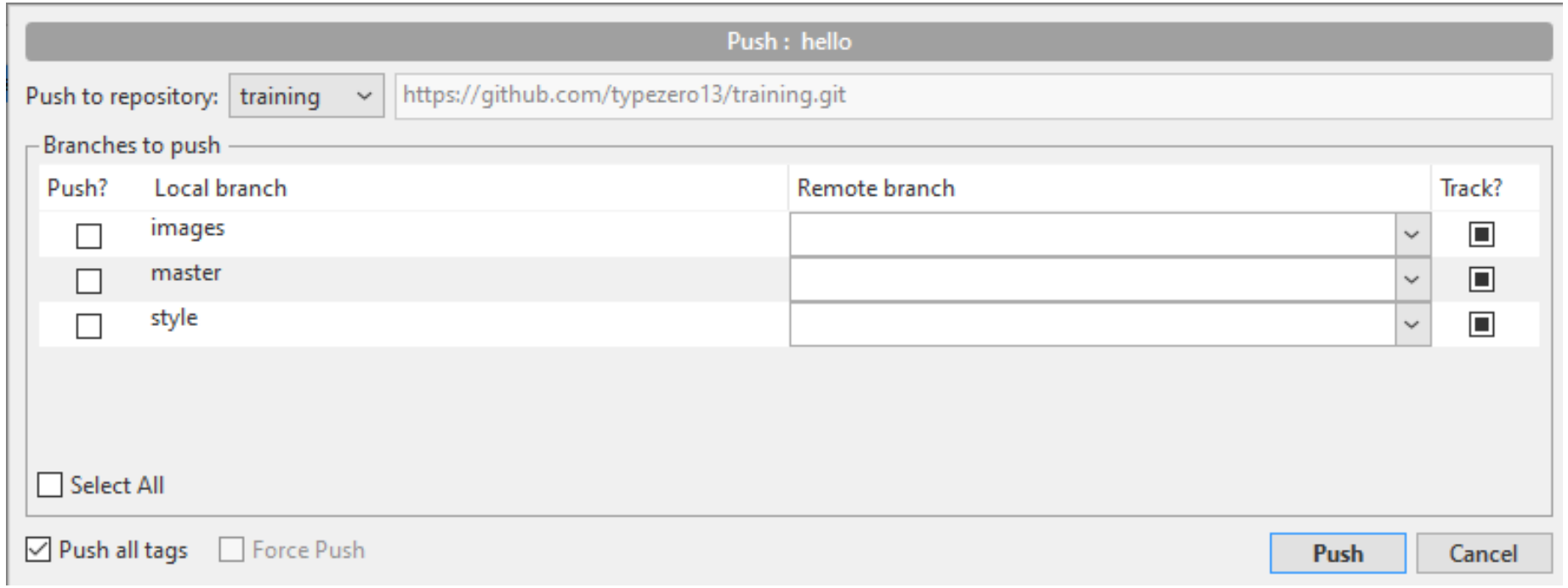
Import code

Github – Add remote repository on SourceTree

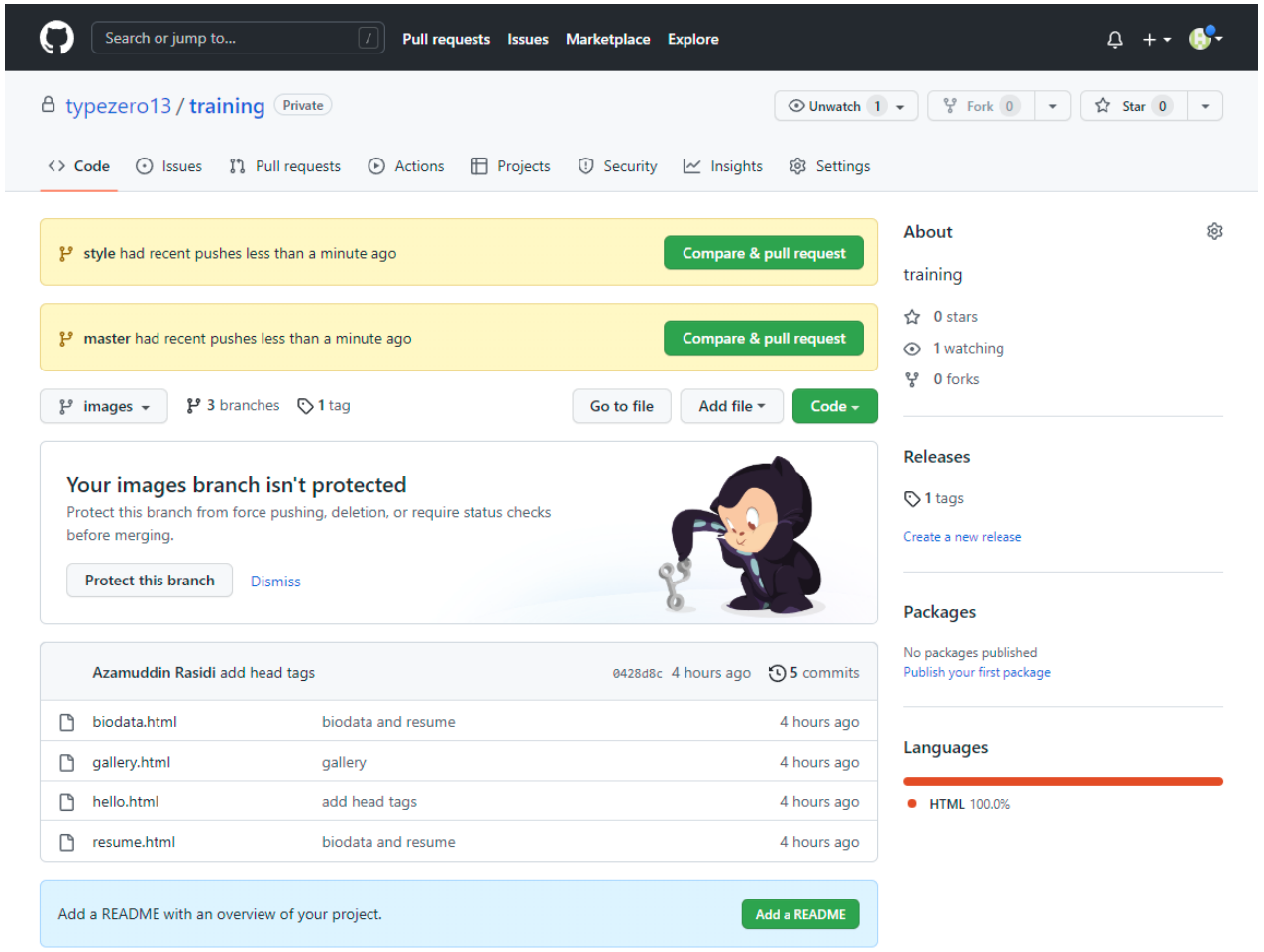




Github – Push to remote branch



Github - Check remote branch



Github Fork

- If you want to contribute to an existing project to which you don't have **push** access, you can **"fork"** the project.
- When you "fork" a project, GitHub will make a copy of the project that is entirely yours, it lives in your namespace, and you can push to it.
- You can fork a project, push to it, and contribute the changes back to the original repository by creating what's called a **Pull Request**.
- To fork a project, visit the project page and click the **"Fork"** button at the top-right of the page.

Github Fork

Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. View existing forks.

Owner * typezero13 / **Repository name** * ionic-conference-app

By default, forks are named the same as their parent repository. You can customize the name to distinguish it further.

Description (optional)

A conference app built with Ionic to demonstrate Ionic

☑ **Copy the** `master` **branch only**

Contribute back to ionic-team/ionic-conference-app by adding your own branch. Learn more.

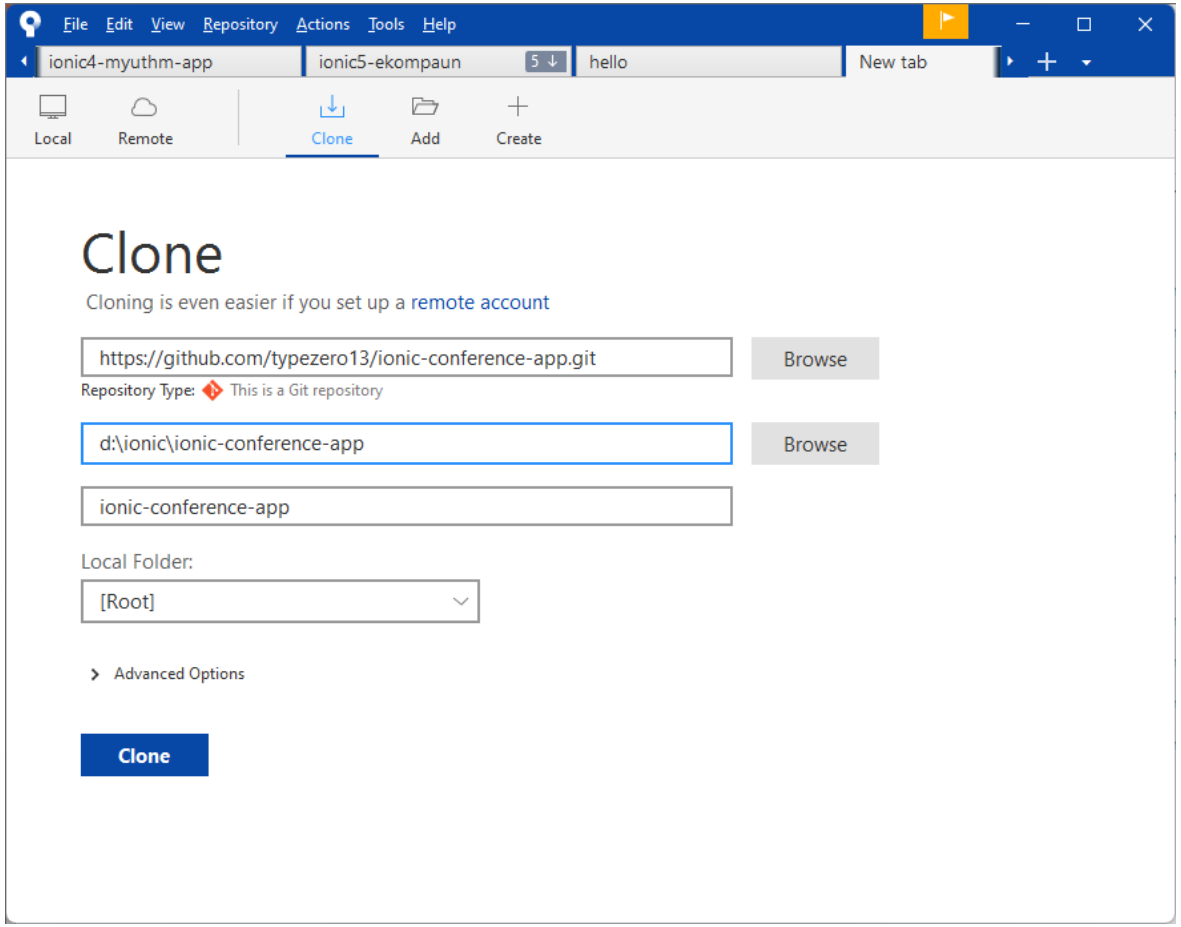
ⓘ You are creating a fork in your personal account.

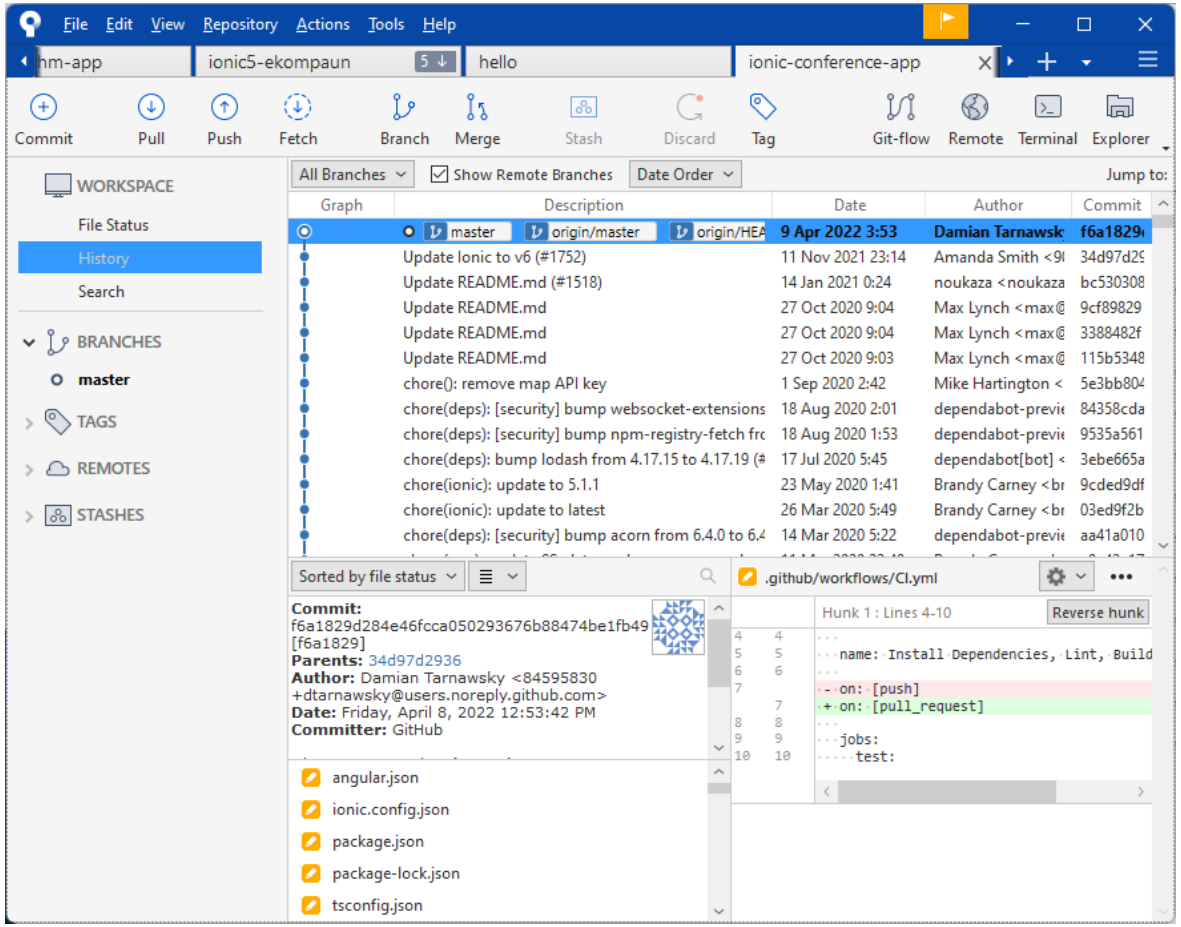
Create fork

GitHub Flow

1. Fork the project.
2. Create a topic branch from master.
3. Make some commits to improve the project.
4. Push this branch to your GitHub project.
5. Open a Pull Request on GitHub.
6. Discuss, and optionally continue committing.
7. The project owner merges or closes the Pull Request.
8. Sync the updated master back to your fork.

GitHub – Clone from remote repository





References

- Chacon, S., Straub, B. (2022). *Pro Git (Everything You Need to Know About Git)* (2nd ed.). Apress.
- Gandhi, R. (2022). *Head First Git: A Learner's Guide to Understanding Git from the Inside Out* (1st ed.). O'Reilly Media.
- Wikipedia
- Reddit

UTHM
Universiti Tun Hussein Onn Malaysia

Thank You