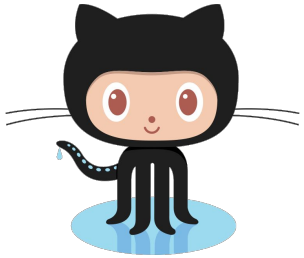


GitHub

Intro To Git Workshop

Presented by Girls Who Code McGill
February 10th, 2022

git

Who are we?
What is our mission?

We are working to

- Close the gender gap in technology.
- Change the image of what a programmer looks like and does.

Meet Our Team

Mohanna Shahrad
Co-President
U2 Computer Science

Alara Ozkutucu
Co-President
U3 Computer Science and Economics

Asu Simla Ayduran
VP Events
U3 Computer Science

Rina Anzarut
VP External
U2 Computer Science
Minor in Psychology

Veronica Xia
VP Communications
Second Year Honours Computer Science and Mathematics

Ying Zhou
VP Internal
U2 Computer Science

Julia Fortin
VP Marketing
U2 Art history

Arsha Misra
VP Finance
U1 Double Concentration in Accounting and Finance

Delal Tomruk
Advisor

Doga Ozkaya
Advisor

Agenda

01 Version Control

02 What is Git?

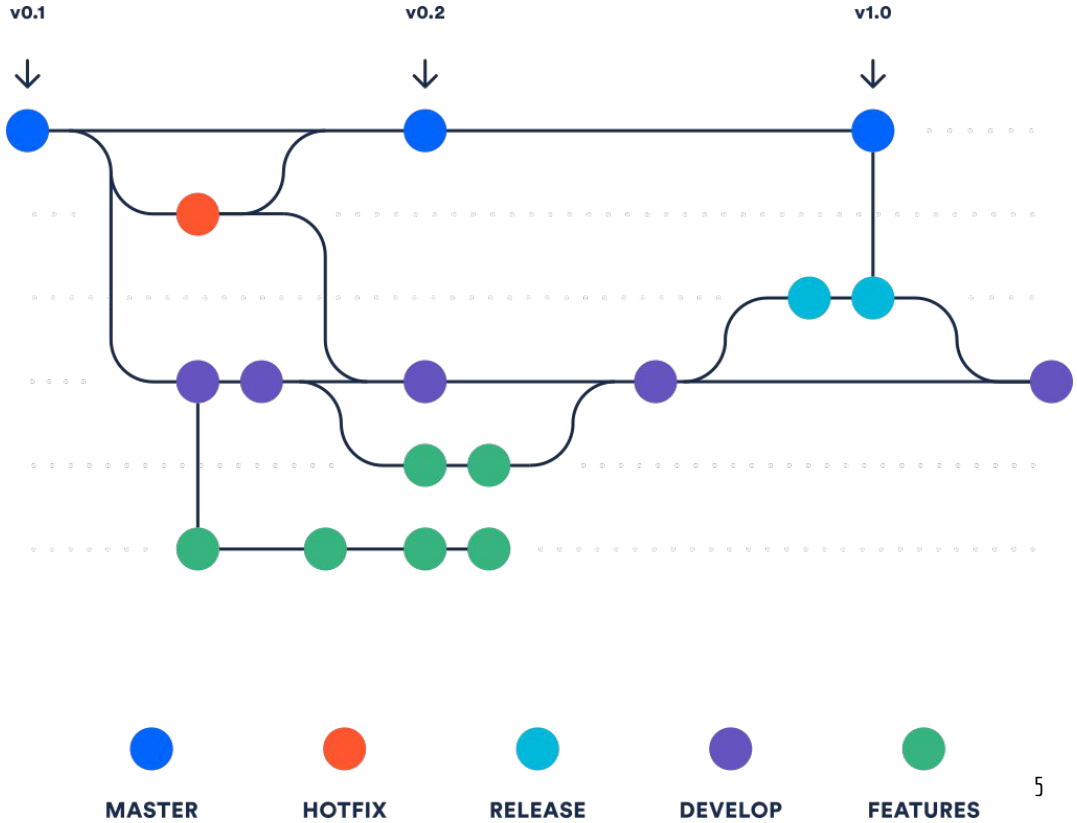
03 Common Commands of Git

04 Demo

05 GitHub Desktop, IDEs Version Control, Resources

What is Version Control?

- Tracking and managing changes to software code (Traceability)
- Helping software teams manage changes to source code over time
- Preserve efficiency and agility as the team scales to include more developers.
- Branching and merging



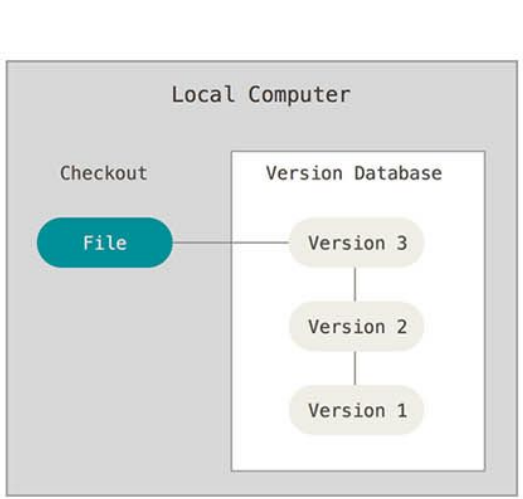
Different Version Control Systems

All developers should access the same file system!

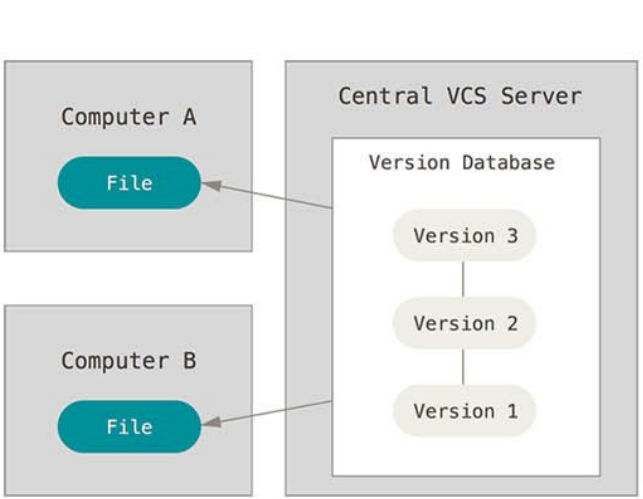
Everyone must have access to the repo via a local network's internet

Everyone works directly with their own local repository.

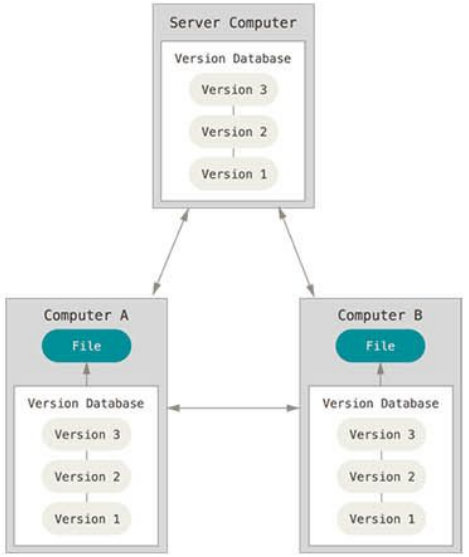
Local Model



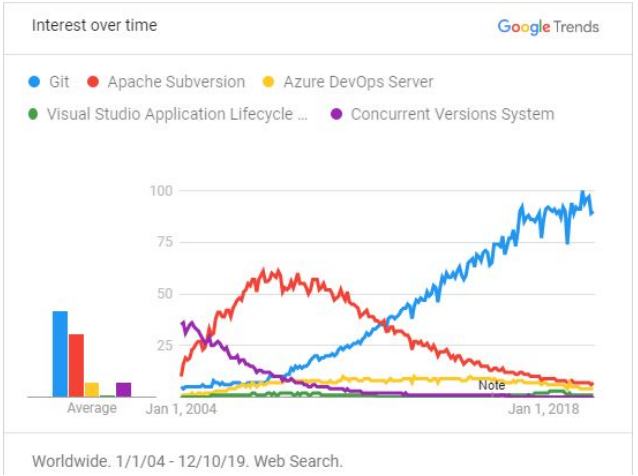
Client-Server Model



Distributed Model

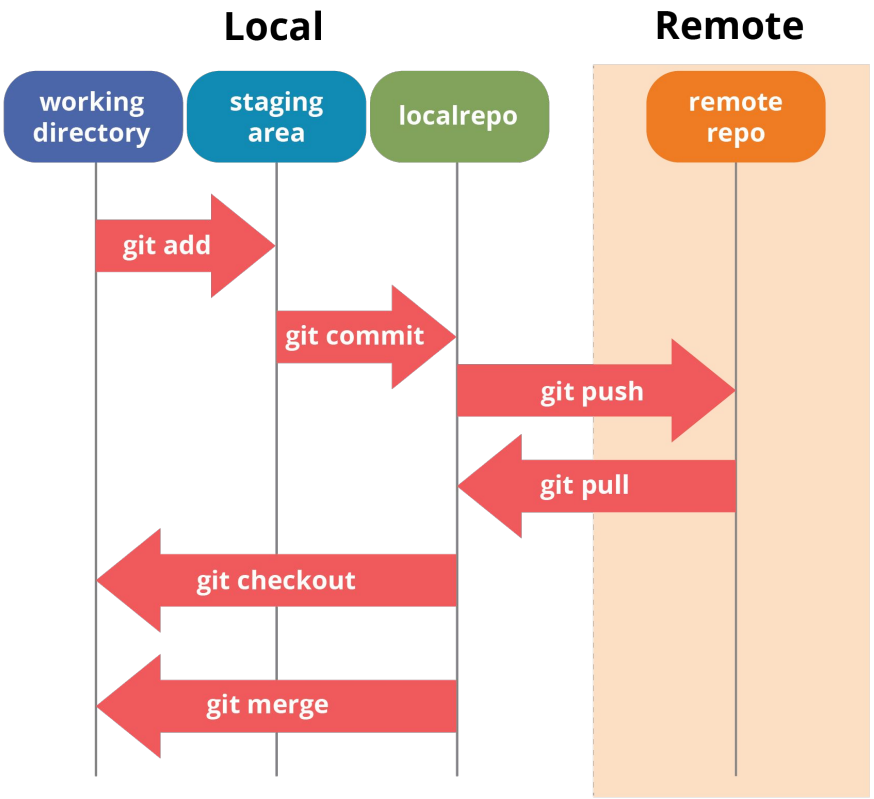


What is Git?



- Most commonly used version control system
- Git runs locally (the history is saved on your computer)
- Possibility of using online hosts (GitHub, Bitbucket, ...)
- Can be used via command-line (terminal) or GUI
- Open source and free
- Git has integrity (everything is checksummed before getting stored)

Git Architecture

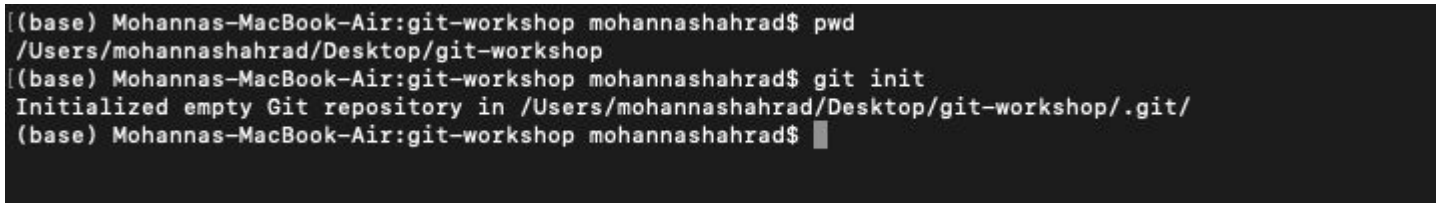


```
16:42 $ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   new_files.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   gulpfile.js

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

        untracked_file.txt
```

Common Git Commands

$ git init

- “git init” turns a directory into an empty Git repository
- First step in creating a git repository



$ git add

- “git add” adds files to the staging area for Git
- Files need to be added before making any commits

```
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$ touch sample.txt
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$ git add sample.txt
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   sample.txt

(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$
```

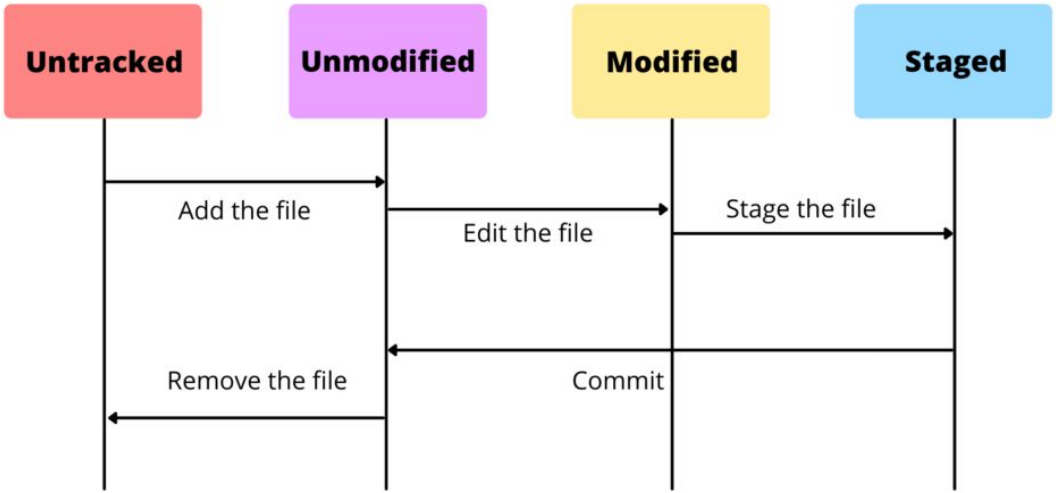
$ git commit

- “git commit” records the changes made to files to our local repository
- Each commit has a unique id
- Best Practice: Add a message to your commits!

```
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$ git commit -m "This is the first commit"
[master (root-commit) d73c087] This is the first commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 sample.txt
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$ git status
On branch master
nothing to commit, working tree clean
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$
```

$ git status

- “git status” shows the current status of the repository



$ git branch

- List all branches (either local or remote): ***git branch -a***
- Adding a branch: ***git branch <branch_name>***
- Deleting a branch: ***git branch -d <branch_name>***

```
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$ git branch feature_1
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$ git branch feature_2
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$ git branch -a
  feature_1
  feature_2
* master
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$ git branch -d feature_2
Deleted branch feature_2 (was d73c087).
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$ git branch -a
  feature_1
* master
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$
```

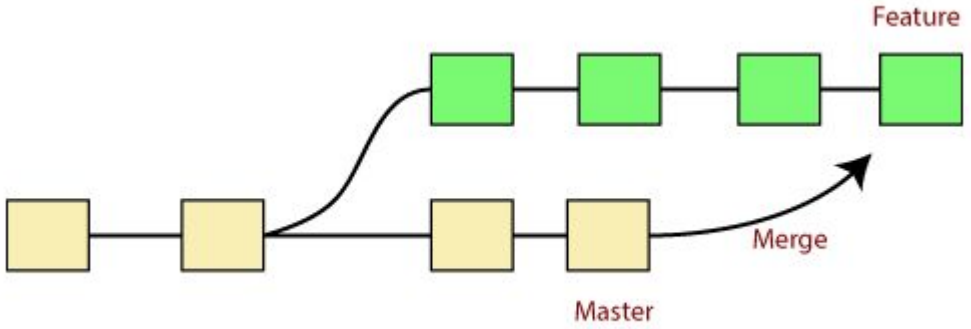
$ git checkout

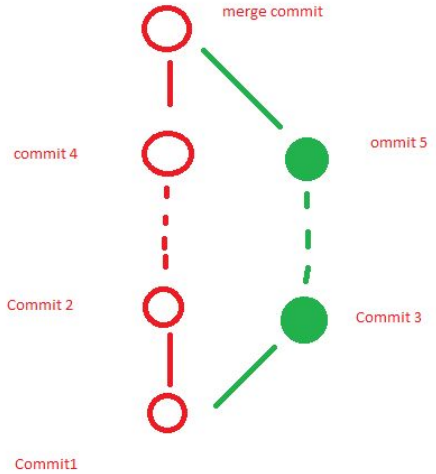
- To switch between branches and change your current working branch
- Use option “-b” if you want to both create and switch to the new branch

```
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$ git branch
  feature_1
* master
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$ git checkout feature_1
Switched to branch 'feature_1'
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$ git branch
* feature_1
  master
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$ git checkout -b feature_2
Switched to a new branch 'feature_2'
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$ git branch
  feature_1
* feature_2
  master
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$
```

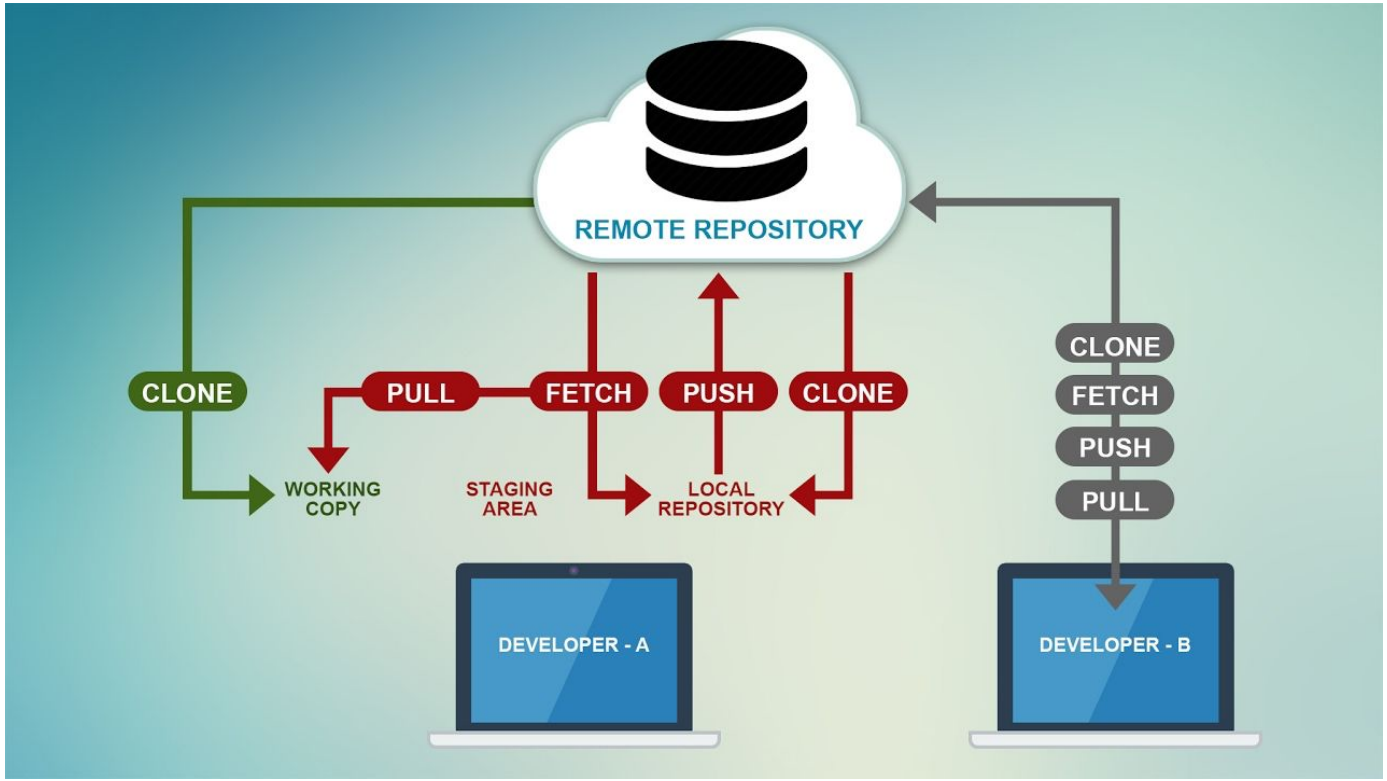
$ git merge

- “git merge” integrates branches together
- It combines the changes in one branch to another branch





Working with remote repositories



$ git remote

- To connect a local repository to a remote one
- You can set a name for the remote repository (to avoid using remote link)
  - Common practice to name the remote repository “origin”
- Option -v lists named remote repositories

```
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$ git remote add origin git@mohannashahrad.git.GWC_McGill_Git_Workshop.com:/mohannashahrad/GWC_McGill_Git_Workshop.git
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$ git remote -v
origin  git@mohannashahrad.git.GWC_McGill_Git_Workshop.com:/mohannashahrad/GWC_McGill_Git_Workshop.git (fetch)
origin  git@mohannashahrad.git.GWC_McGill_Git_Workshop.com:/mohannashahrad/GWC_McGill_Git_Workshop.git (push)
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$
```

$ git clone

- To create a local working copy of an existing remote repository
- Same as “git init” when working with a remote repository

```
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$ git clone https://github.com/mohannashahrad/GWC_McGill_Git_Workshop.git
Cloning into 'GWC_McGill_Git_Workshop'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
(base) Mohannas-MacBook-Air:git-workshop mohannashahrad$
```

$ git pull

- To get the latest version of a remote repository
- ***git pull <branch_name> <remote_URL/remote_name>***

```
(base) Mohannas-MacBook-Air:GWC_McGill_Git_Workshop mohannashahrad$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/mohannashahrad/GWC_McGill_Git_Workshop
   6f7d290..23d14d0  main       -> origin/main
Updating 6f7d290..23d14d0
Fast-forward
 README.md | 4 +++-
 1 file changed, 3 insertions(+), 1 deletion(-)
(base) Mohannas-MacBook-Air:GWC_McGill_Git_Workshop mohannashahrad$ cat README.md
# GWC_McGill_Git_Workshop

This is a test repository for the Intro to Git Workshop presented by Girls Who Code McGill.
(base) Mohannas-MacBook-Air:GWC_McGill_Git_Workshop mohannashahrad$
```

$ git push

- To send the local commits to the remote repository
- ***git push <remote_URL/remote_name> <branch>***

```
(base) Mohannas-MacBook-Air:GWC_McGill_Git_Workshop mohannashahrad$ ls
README.md       sample.txt
(base) Mohannas-MacBook-Air:GWC_McGill_Git_Workshop mohannashahrad$ cat sample.txt
This is only for demo purposes.
(base) Mohannas-MacBook-Air:GWC_McGill_Git_Workshop mohannashahrad$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
(base) Mohannas-MacBook-Air:GWC_McGill_Git_Workshop mohannashahrad$ git push
Username for 'https://github.com': mohannashahrad
Password for 'https://mohannashahrad@github.com':
remote: Support for password authentication was removed on August 13, 2021. Please use a personal access token instead.
remote: Please see https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/ for more information.
fatal: Authentication failed for 'https://github.com/mohannashahrad/GWC_McGill_Git_Workshop.git/'
(base) Mohannas-MacBook-Air:GWC_McGill_Git_Workshop mohannashahrad$
```

GitHub Personal Access Token

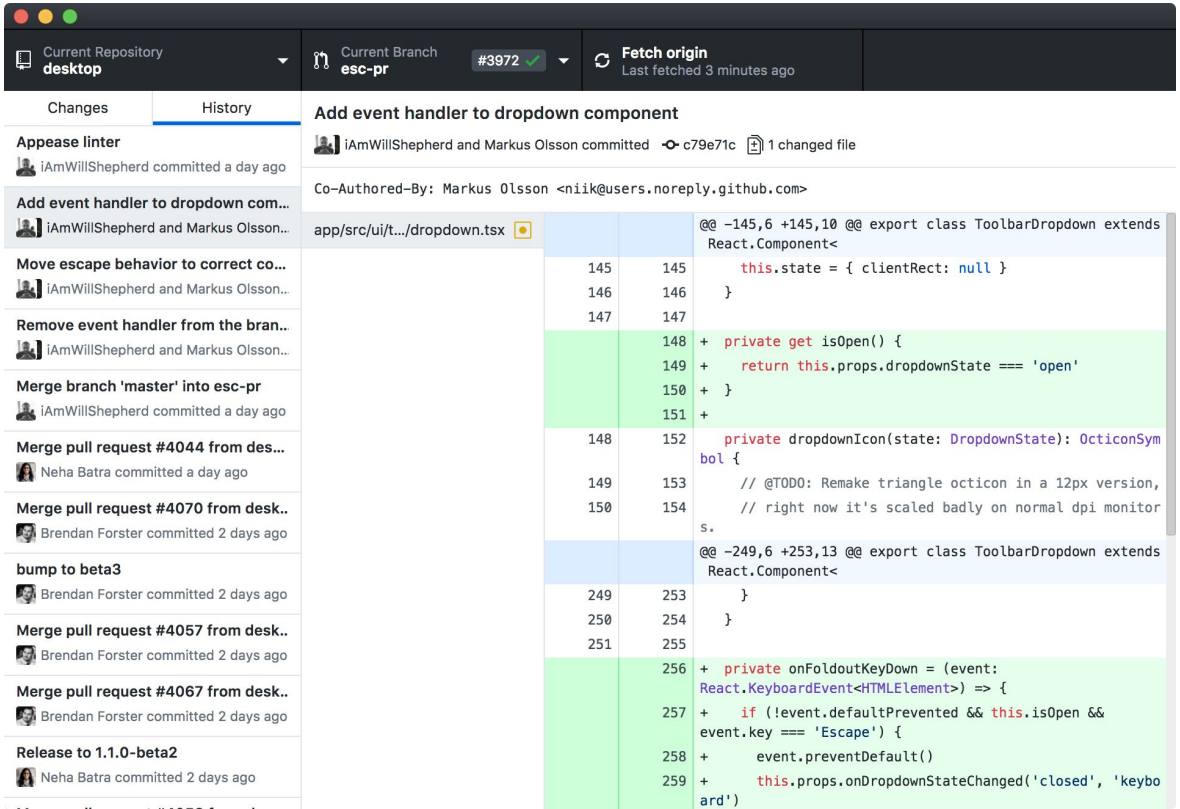
• When you use git push for the first time, it will prompt you for your GitHub username and password. However, your password will not work! (This was a very recent change).
• Instead, for security, when linking directly with git, you must generate a Personal Access Token and use that as your password.
• You can find these in GitHub: Settings -> Developer Settings -> Personal Access Tokens -> Generate new token • Hit generate, but be sure to check the “repo” box. This says that you authorize connections made with this access token to modify GitHub repositories.
• Copy this token and paste it (as you’d normally do – cmd-V or control-V) into the password prompt that comes up when you do git push, and hit enter.

How to Setup?

- Verify if Git is installed:
- For Mac or Linux, Git should be installed: verify by typing “git --version” in your terminal (no brackets when you copy paste)
- If not installed, follow this link: https://www.atlassian.com/git/tutorials/install-git
- For Windows, it is easier with GitBash
- For Mac, Homebrew is recommended
- Install a Code Editor
- VSCode https://code.visualstudio.com/

DEMO

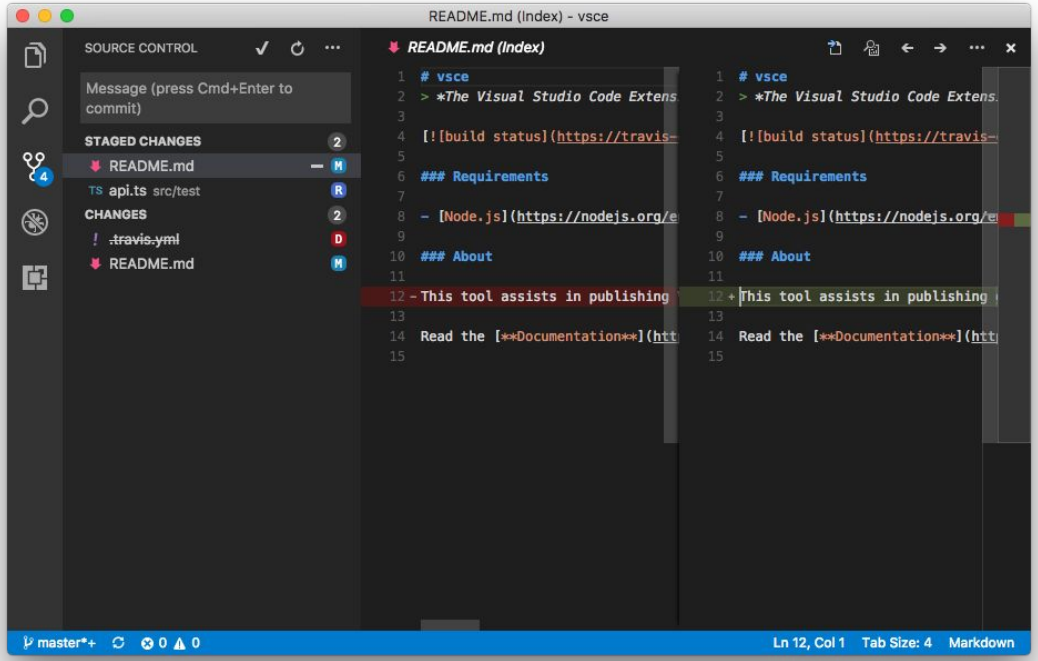
GitHub Desktop



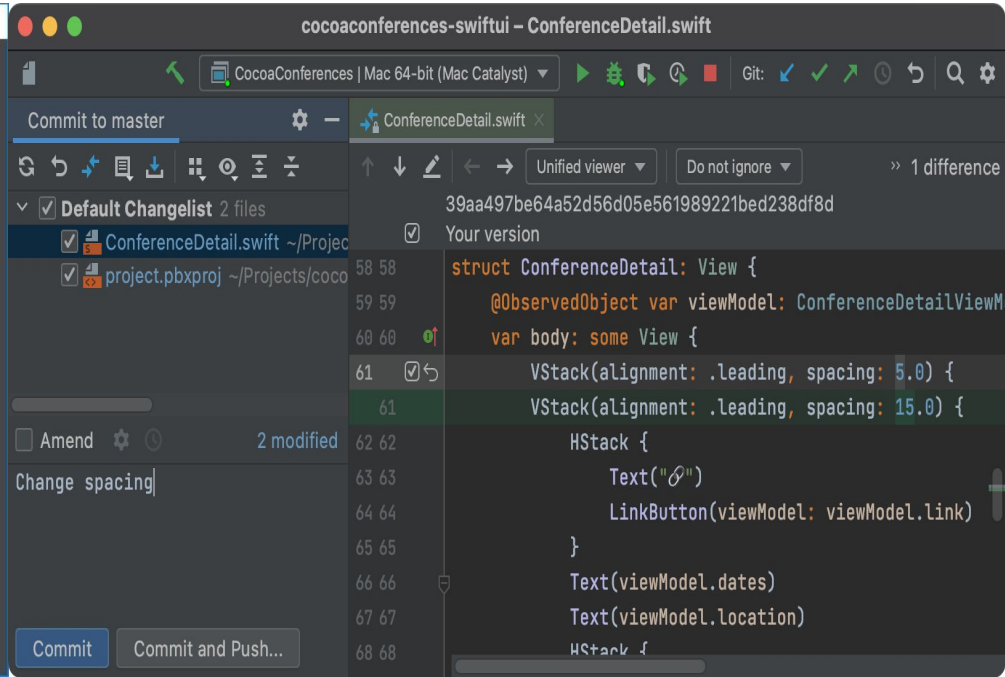
- Attribute commits with collaborators easily
- Checkout branches with pull requests and view CI statuses
- Syntax highlighted diffs

Click here to install!

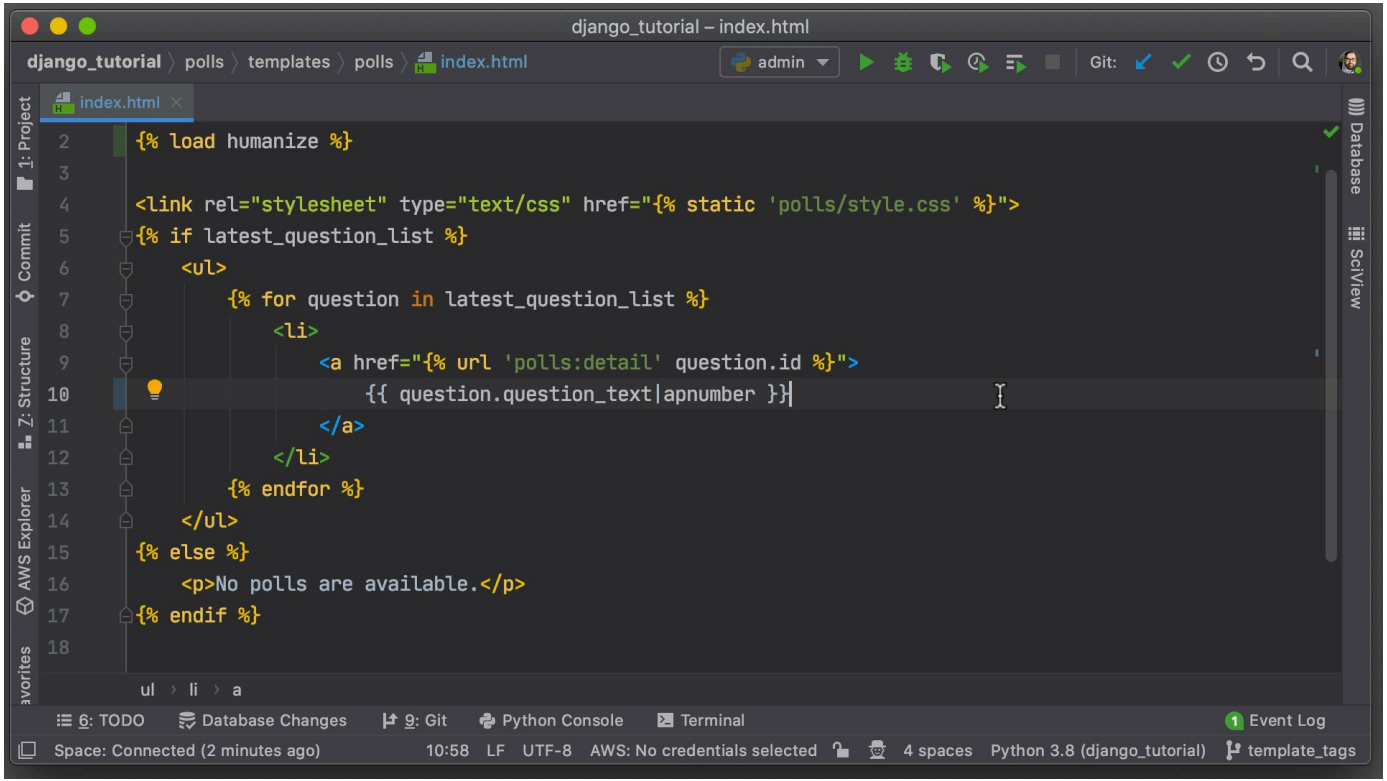
Version Control in IDEs - VSCode



Version Control in IDEs - IntelliJ



Version Control in IDEs - PyCharm



[Git and GitHub learning resources]

[Pro Git EBook]

Git Cheat Sheet

GIT BASICS

`git init <directory>`	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
`git clone <repo>`	Clone repo located at `<repo>` onto local machine. Original repo can be located on the local filesystem or on a remote machine via `HTTP` or `SSH`.
`git config user.name <name>`	Define author name to be used for all commits in current repo. Devs commonly use `--global` flag to set config options for current user.
`git add <directory>`	Stage all changes in `<directory>` for the next commit. Replace `<directory>` with a `<file>` to change a specific file.
`git commit -m "<message>"`	Commit the staged snapshot, but instead of launching a text editor, use `<message>` as the commit message.
`git status`	List which files are staged, unstaged, and untracked.
`git log`	Display the entire commit history using the default format. For customization see additional options.
`git diff`	Show unstaged changes between your index and working directory.

UNDOING CHANGES

`git revert <commit>`	Create new commit that undoes all of the changes made in `<commit>`, then apply it to the current branch.
`git reset <file>`	Remove `<file>` from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.
`git clean -n`	Shows which files would be removed from working directory. Use the `-f` flag in place of the `-n` flag to execute the clean.

REWRITING GIT HISTORY

`git commit --amend`	Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.
`git rebase <base>`	Rebase the current branch onto `<base>`. `<base>` can be a commit ID, branch name, a tag, or a relative reference to `HEAD`.
`git reflog`	Show a log of changes to the local repository's `HEAD`. Add `--relative-date` flag to show date info or `--all` to show all refs.

GIT BRANCHES

`git branch`	List all of the branches in your repo. Add a `<branch>` argument to create a new branch with the name `<branch>`.
`git checkout -b <branch>`	Create and check out a new branch named `<branch>`. Drop the `-b` flag to checkout an existing branch.
`git merge <branch>`	Merge `<branch>` into the current branch.

REMOTE REPOSITORIES

`git remote add <name> <url>`	Create a new connection to a remote repo. After adding a remote, you can use `<name>` as a shortcut for `<url>` in other commands.
`git fetch <remote> <branch>`	Fetches a specific `<branch>`, from the repo. Leave off `<branch>` to fetch all remote refs.
`git pull <remote>`	Fetch the specified remote's copy of current branch and immediately merge it into the local copy.
`git push <remote> <branch>`	Push the branch to `<remote>`, along with necessary commits and objects. Creates named branch in the remote repo if it doesn't exist.

ATLASSIAN

Visit atlassian.com/git for more information, training, and tutorials

Additional Options +

GIT CONFIG

`git config --global user.name <name>`	Define the author name to be used for all commits by the current user.
`git config --global user.email <email>`	Define the author email to be used for all commits by the current user.
`git config --global alias. <alias-name> <git-command>`	Create shortcut for a Git command. E.g. `alias.glog "log --graph --oneline"` will set `"git glog"` equivalent to `"git log --graph --oneline.`
`git config --system core.editor <editor>`	Set text editor used by commands for all users on the machine. `<editor>` arg should be the command that launches the desired editor (e.g., vi).
`git config --global --edit`	Open the global configuration file in a text editor for manual editing.

GIT LOG

`git log -<limit>`	Limit number of commits by `<limit>`. E.g. `"git log -5"` will limit to 5 commits.
`git log --oneline`	Condense each commit to a single line.
`git log -p`	Display the full diff of each commit.
`git log --stat`	Include which files were altered and the relative number of lines that were added or deleted from each of them.
`git log --author= "<pattern>"`	Search for commits by a particular author.
`git log --grep="<pattern>"`	Search for commits with a commit message that matches `<pattern>`.
`git log <since>..<until>`	Show commits that occur between `<since>` and `<until>`. Args can be a commit ID, branch name, `HEAD`, or any other kind of revision reference.
`git log -- <file>`	Only display commits that have the specified file.
`git log --graph --decorate`	`--graph` flag draws a text based graph of commits on left side of commit msgs. `--decorate` adds names of branches or tags of commits shown.

GIT DIFF

`git diff HEAD`	Show difference between working directory and last commit.
`git diff --cached`	Show difference between staged changes and last commit

GIT RESET

`git reset`	Reset staging area to match most recent commit, but leave the working directory unchanged.
`git reset --hard`	Reset staging area and working directory to match most recent commit and **overwrites all changes** in the working directory.
`git reset <commit>`	Move the current branch tip backward to `<commit>`, reset the staging area to match, but leave the working directory alone.
`git reset --hard <commit>`	Same as previous, but resets both the staging area & working directory to match. **Deletes** uncommitted changes, and **all commits after** `<commit>`.

GIT REBASE

`git rebase -i <base>`	Interactively rebase current branch onto `<base>`. Launches editor to enter commands for how each commit will be transferred to the new base.

GIT PULL

`git pull --rebase <remote>`	Fetch the remote's copy of current branch and rebases it into the local copy. Uses git rebase instead of merge to integrate the branches.

GIT PUSH

`git push <remote> --force`	Forces the `git push` even if it results in a non-fast-forward merge. Do not use the `--force` flag unless you're absolutely sure you know what you're doing.
`git push <remote> --all`	Push all of your local branches to the specified remote.
`git push <remote> --tags`	Tags aren't automatically pushed when you push a branch or use the `--all` flag. The `--tags` flag sends all of your local tags to the remote repo.

Resources

- https://www.atlassian.com/git/tutorials/
- http://guides.beanstalkapp.com/version-control/
- https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F
- https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control
- https://education.github.com/git-cheat-sheet-education.pdf

Thank You!

Any questions?

 girlswhocodemcgill@gmail.com

 @girlswhocodemcgill

 @girls-who-code-mcgill