Git Worktrees

ADVANCED GIT

Amanda Crawford-Adamo
Software and Data Engineer

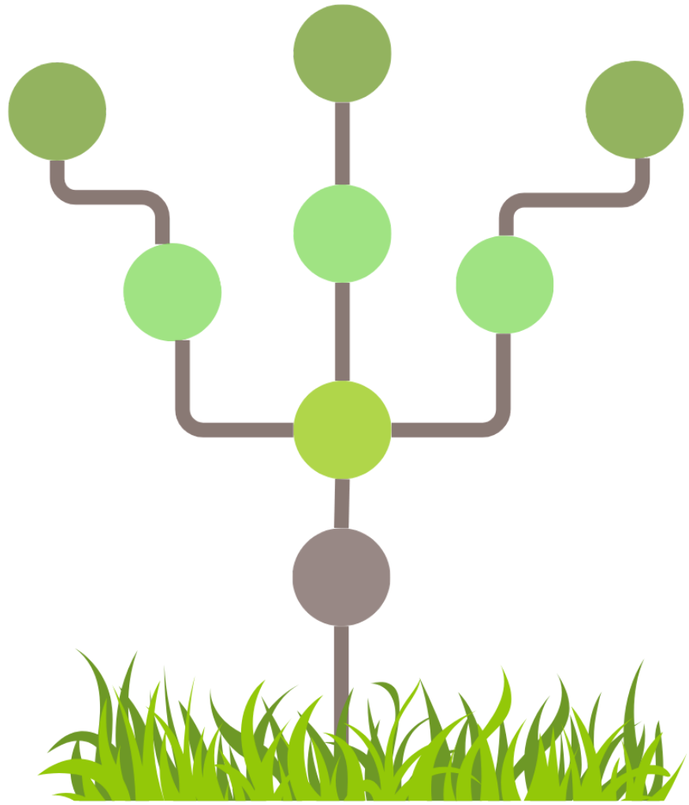
datacamp

What is a Git Worktree?

Git Worktree Command

```
git worktree
```

- Can *"checkout"* multiple branches in your workspace.
- Similar to a repo checkout, but efficient
- No need for stashing changes
- No need to switch between branches during development



Git Worktree versus Git Switch

This tables compares using git worktree vs git switch in a **development workflow.**

Git Worktree	Git Switch
Multiple active branches	One active branch at a time
Separate directories	Single working directory
No need to stash changes	May require stashing

Creating a Git Worktree

Create new work tree from <branch> into directory <path>

```
git worktree add <path> <branch>
```

Example

Create a new work tree from the `bugfix/data-validation` branch into the `../etl-bugfix` directory

```
git worktree add ../etl-bugfix bugfix/data-validation
```

Listing and Removing Worktrees

- Lists all active worktrees: `git worktree list`

Example Output

```
$ git worktree list
flight-pipeline             a1b2c3d [main]
flight-pipeline-feature     e4f5g6h [feature]
flight-pipeline-hotfix      i7j8k9l [hotfix]
```

- Removes a worktree from a <path>: `git worktree remove <path>`

Example Output

```
$ git worktree remove flight-pipeline-hotfix
flight-pipeline-hotfix: deleted
```

When to use Git Worktrees

When to use:

- Working on multiple features simultaneously
- Handling urgent bug fixes without disrupting ongoing work
- Running tests on different branches in parallel
- Code reviews while continuing development

Reconsider when:

- Disk space is limited
- Projects with frequent updates and complex merge

Best practices for Git Worktrees

When using Git worktrees, keep these tips in mind:

1. Use clear naming conventions for worktree directories
2. Regularly prune unused worktrees to keep the workspace clean
3. Be mindful of disk space, especially with large projects
4. Use worktrees for short-lived parallel work to avoid confusion

Let's practice!

ADVANCED GIT

Git Submodules

ADVANCED GIT

Amanda Crawford-Adamo
Software and Data Engineer

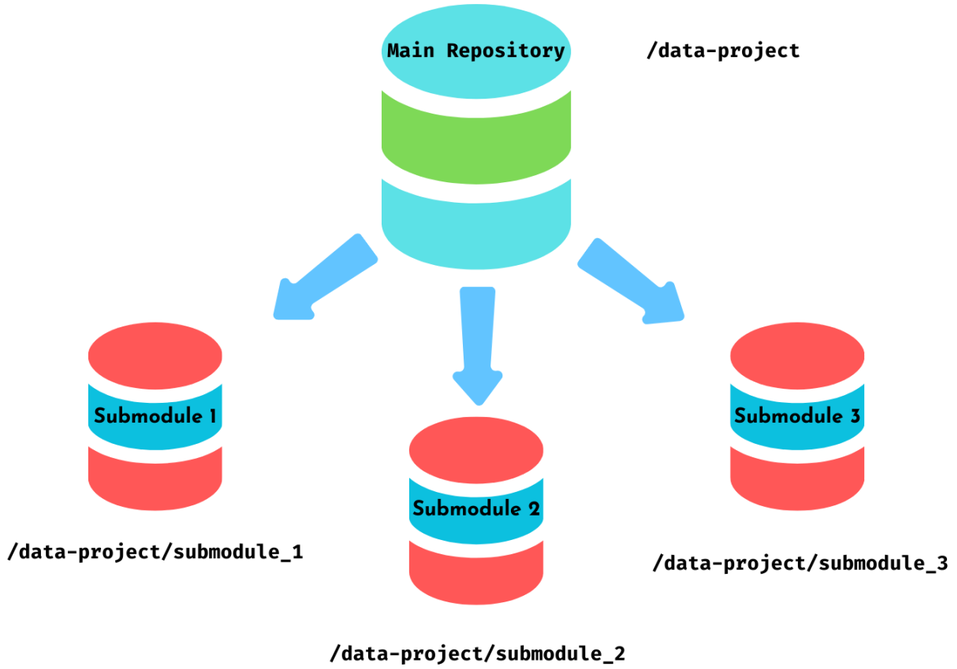
datacamp

What is a Git Submodule?

Git Submodule

```
git submodule
```

- A repository nested within another repository
- Separate version control and history
- Submodule changes does not affect main repo
- Main repo can reference a specific version of a submodule



Adding a submodule

Adding a submodule using the link or directory under path folder.

```
git submodule add <repository link|dir> <path>
```

Example

Adds the data validator library to the ETL project under the `libs/validator` folder in the ETL repo.

```
git submodule add https://github.com/example/data-validator.git libs/validator
```

Listing submodules

List all submodules in a project

```
git submodule status
```

Example

```
$ git submodule status
e1f2...7w8x9 data_cleaning_lib
a1b2...q7r8 api_connector
d9e8...t3u2 visualization_toolkit
```

Updating submodules

Update submodule with the latest changes

There are several options:

1. Updates all submodules where the source code is on your **local** computer.

```
git submodule update --init
```

2. Updates all submodule where the source code is on a **remote** repo.

```
git submodule update --init --remote
```

3. Updates a specific submodules

```
git submodule update --init <path_to_submodule>
```

Removing submodules

Remove a submodule process

1. Deinitialize the submodule.

```
git submodule deinit <submodule_name>
```

2. Remove the submodule from git repo index.

```
git rm <path>
```

Extracting a submodule from a large repo

1. Copy all files that need to be in the new submodule repo into another folder outside the repo.
2. Inside the new folder, create a new repository for the submodule:

```
git init <new-submodule>
```

3. Use git **filter-repo** to extract the relevant files and history from the main project:

```
git filter-repo --path <extract_path> --invert-paths
```

4. Add the extracted repository as a **submodule** to the main project:

```
git submodule add <new-submodule_path> <path_to_store_submodule>
```

When to use submodules and best practices

Use cases:

1. Managing external libraries
2. Sharing code across projects
3. Maintaining specific versions of dependencies

Best practices:

1. Keep submodules updated
2. Use relative paths
3. Communicate changes with team

Let's practice!

ADVANCED GIT

Git Large File Storage

ADVANCED GIT

Amanda Crawford-Adamo
Software and Data Engineer

What is Git Large File System?

Git LFS Command:

```
git lfs
```

- Git LFS: Git Large File Storage
- Replace large files in repo
- Small pointer files
- Large files separate from repo

Benefits:

1. Reduced repository size
2. Faster cloning and fetching
3. Efficient binary file handling
4. Improved collaboration on large files

Git LFS initialization process

- Initialize Git LFS

```
git lfs install
```

- Setup files to track and generate `.gitattributes` file

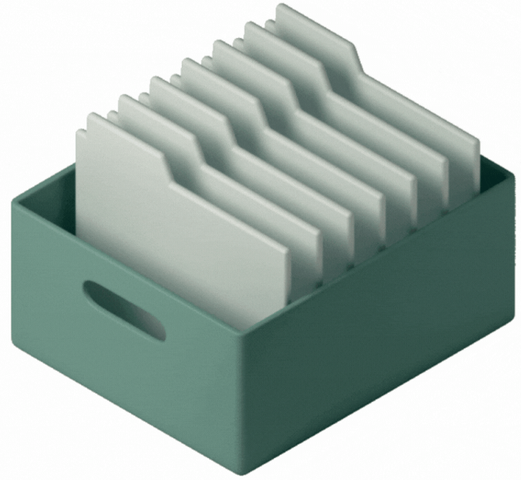
```
git lfs track "*.csv"
```

- Add to git index `.gitattributes` with tracking config

```
git add .gitattributes
```

- Commit new changes

```
git commit -m "Track CSV files"
```



Git LFS update process

1. Add new file using `git add`

```
git add large_file.csv
```

2. Commit and push the changes

```
git commit -m "Update large CSV file"
git push origin main
```

3. Download changes

```
git pull
git lfs pull  # If needed to explicitly download LFS content
```

When to use Git LFS

When to use:

1. Need to track changes to large datasets (CSV, JSON, etc.)
2. Machine learning models
3. Binary assets (images, videos)
4. Version control compressed or installer files

When not to use:

1. Infrequently updated large files
2. Small text files, like code
3. Tight storage quotas

Best practices

1. Efficient large file management
2. Improved collaboration on data-heavy projects
3. Seamless integration with Git workflow

Tips:

1. Track files selectively
2. keep your team informed about LFS usage
3. Regularly prune LFS cache

Let's practice!

ADVANCED GIT

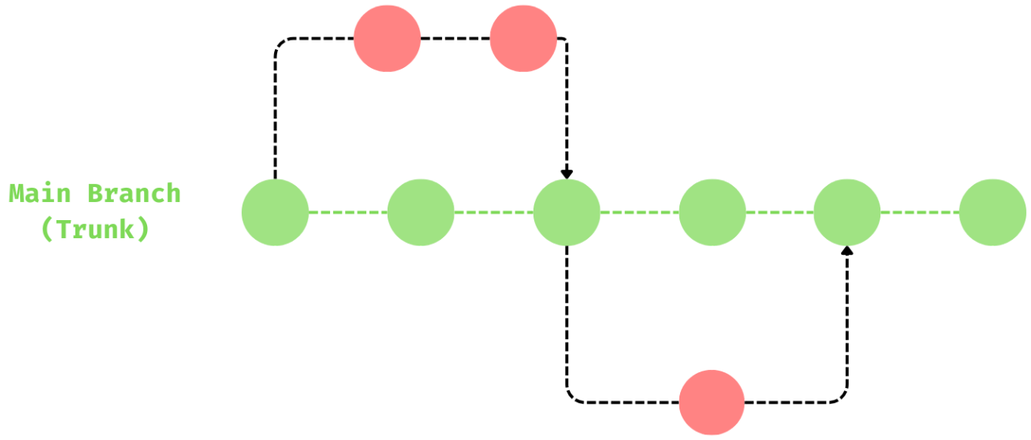
Trunk Based Development

ADVANCED GIT

Amanda Crawford-Adamo
Software and Data Engineer

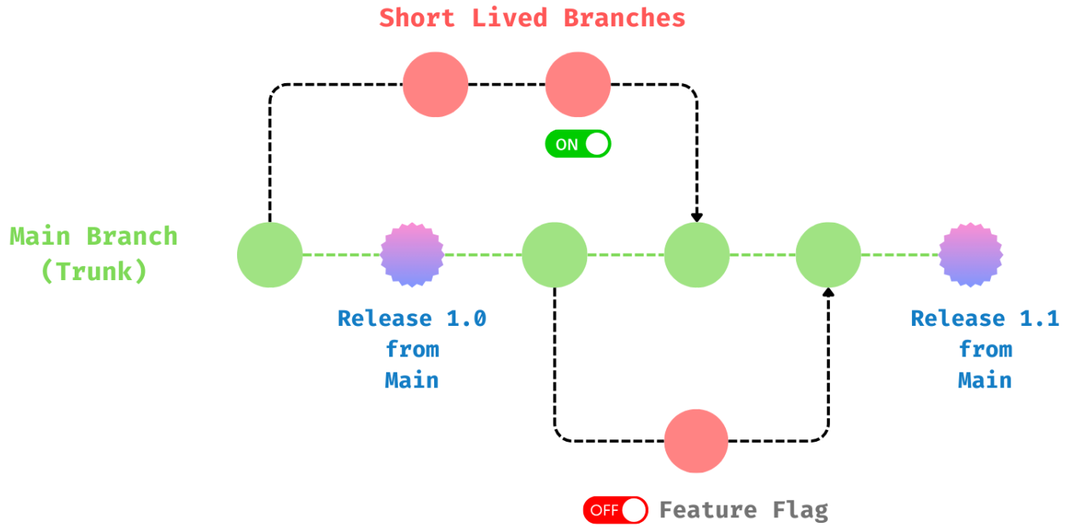
What is Trunk Based Development?

- Source control branching CI/CD model
- Developer no longer push to separate release branches
- Changes are from short-lived branches pushed to main
- Small and frequent updates



Core principles of Trunk Based Development

1. Frequent commits to main
2. Short-lived feature branches (< 1 day)
3. Continuous integration
4. Feature flags for incomplete work



Feature flagging in TBD

- Manages incomplete features
- Prevent user from being affected
- Features gradually released

Example Feature Flag Code

```
if feature_flag_enabled('new_feature'):
    # New feature code
else:
    # Old feature code
```

Continuous integration in TBD

- Commits to main trigger automated build **and** tests
- Reduce maintenance and faster releases
- Product alway reliable and stable
- Ensure secure code and compliance with industry standards
- Maintain code quality and reduce bug risk

Benefits and challenges of TBD

Benefits:

- Reduced merge conflicts
- Faster release cycles
- Improved code quality
- Better collaboration

Challenges:

- Requires team discipline
- Needs robust testing
- Initial learning curve
- Managing incomplete features

Best Practices

1. Commit small changes frequently
2. Automate testing and deployment
3. Use feature flags for incomplete work
4. Conduct regular code reviews
5. Monitor after deployment

Let's practice!

ADVANCED GIT

Wrap Up

ADVANCED GIT

Amanda Crawford-Adamo
Software and Data Engineer

Chapter Learnings

Chapter 1 **Advanced Merging Strategies**	Chapter 2 **Git History and Exploration**	Chapter 3 **Advanced Repository Management**
Fast Forward Recursive Squash Octopus Rebase	Cherry Picking Bisect Filter-Repo Reflog	Worktrees Submodules Git Large File Storage Trunk Based Development

Key Takeaways

Manage complex merge and commit history

Handle large files and organize code efficiently

Investigate and modify repository history

Implement modern dev workflows like Trunk Based Development

Next Steps

- Learn about git hooks
- Explore topics on advanced CI/CD integration techniques
- Apply your skills by contributing to open-source projects
- Keep an update on new Git features and updates

Congratulations!

ADVANCED GIT