Git Version Control System

Topics

- Workflow
- Initilization
- Branching
- Adding and Commiting Files
- Merging
- Pulling and Pushing
- Status
- Non-Tracked Files
- Topics of Further Interest

Workflow

- Untracked
  - These are not yet in the repository
  - `add` or *.gitignore*
- Unmodified
  - These have not changed since the current head
- Modified
  - These have change since the current head
  - `add`
- Staged
  - These are waiting to be committed
  - `commit`
- *Remote*
  - Files at a remote repository's head
  - `pull` or `push`

Workflow

1. File starts untracked
   - git understands that the file is there
   - git will not do anything about the file being there
2. File is unmodified
   - git will watch for the file to change
3. File is modified
   - git sees the file has changed and awaits it being staged

A basic workflow:

1. Initialize the repository
2. Add existing items to the repository
3. Commit the existing items to the repository with an appropriate
4. Create a new branch
5. Checkout the new branch
6. Modify contents of the branch
7. Merge branch back into master
8. Repeat from step 4

A basic workflow with a remote repository:

1. Clone the previously existing repository
2. Create a new branch
3. Checkout the new branch
4. Modify the contents of the branch
5. Checkout the master branch
6. Pull from the remote repository
7. Merge from your branch to the master branch
8. Pull from the remote repository
9. Push to the remote repository
10. Repeat from step 2

Initialization

`init`

Usage:

1. `git init`
   - Does the background work to start a git repository in the current directory which has no files currently tracked.

Initialization

`clone`

Usage:

1. git clone *remote_repository_address*
   - Creates a new git repository in the current directory that is a copy of the remote repository given.

Note: This will create another directory below your current directory for the repository.

Branching

`branch`

Usage:

1. git branch
   - review the branches in the repository
2. git branch *new_branch_name*
   - create a new branch
3. git branch -d *new_branch_name*
   - delete the branch

Note: Creating a branch does not change your current branch.

Branching

`checkout`

Usage:

1. git checkout *branch_name*
   - Change to the branch
2. git checkout *commit*
   - Move the current state of your copy to a particular commit

add and commit

add

Usage:

1. git add *file* . . .
   - Add stage one or more files

commit

Usage:

1. git commit -m “Commit Message”
   - The -m flag is optional: without it the result drops you into the default editor to create the message

Merging

`merge`

Usage:

1. git merge *branch_name*
   - Merges the named branch into the **current** branch

How to Resolve Conflicts

1. Open a file with a conflict
2. Find the conflict
   - Conflicts are begun with <<<<<<< yours:name_of_file
   - Differences are seperated by =======
   - Conflicts end with >>>>>>> theirs:name_of_file
3. Remove the markers and choose the lines of code which should not be in the result of the merge.
4. Save the file.
5. Repeat from 1 until there are not more conflicts.
6. Add and commit the results of the merge.

Pull and Push

`pull`

Usage:

1. git pull [remote_repository] [branch_name]
   - The two parameters are optional if the defaults are set.

`push`

Usage:

1. git push [remote_repository]
   - This pushes all the updated branches to their equivalent in the remote repository.

Status

`status`

Usage:

1. git status -b
   - This gives the status of files in the repository.
   - The -b flag also provides the name of the current branch

File Statuses

- untracked
- unmodified
- modified
- staged

Non-Tracked Files

- Files that you do not wish to be part of your repository
  - Typically generated artifacts such as executables
- These files are typically handled via a file called *.gitignore*
  - More than one ignore can be used in a single repository
- This removes the unwanted files from being marked as untracked

Non-Tracked Files

Format

- Any line starting with a hash (#) is a comment
- Empty lines match nothing
- All file paths should be relative to the ignore file
- Shell globbing patterns are used
  - i.e. *.exe -> ignore all executable files
- A line starting with ! whitelists the pattern
- To specify a directory end with a /
  - e.g. foo is a file; foo/ is a directory

Topics of Further Interest

- `log`
  - gives a history of commits
  - particularly useful for finding the hash to checkout a particular commit
- `remote`
  - setup remote knowledge of remote repositories
- `stash`
  - Similar to the concept of shelving in other VCS
- `rebase`
  - A method for changing how branches are related
- `diff`
  - Show differences between two states of the repository

More Topics of Further Interest

- `fetch`
  - Get the changes but do not integrate
- `reset`
  - Move the current head
- `tag`
  - Mark git objects
- `mv`
  - Move a files location within a repository
- `rm`
  - Stop tracking the changes to a file