Git

Prajval

Contents

1. Git Worktree

Manage multiple working trees attached to the same repository. This allows to checkout multiple branches at a time.
A repository has one main worktree(if it's not a bare repository) and zero multiple linked worktrees.

add

Add a new working tree. This new worktree is called an linked worktree as opposed to the main worktree prepared by the git-init or git-clone.

```
git worktree add <path> [<commit-ish>]

<path> - directory where the checkout should be.
<commit-ish> - branch or commit id
```

remove

remove a worktree using this option.

```
git worktree remove
```

list

list details of each worktree

```
git worktree list
```

1.2 Workflow using bare repository

Advantage in using a bare repo: all the checkout can be inside the repo itself.

1. init or clone with –bare

   ```
   git clone --bare <upstream>
   ```

2. add worktrees with banch name or commit id

   ```
   git worktree add master
   git worktree add foo
   ```

3. do changes in any of the worktrees
4. remove worktrees which are not required

   ```
   git remove foo
   ```

1.2.1 gitworktree neovim plugin

- Provides keybinds and Telescope support.
- when swithing branches/worktrees same files will be open in the editor.
- status line support.

plugin installation

1. Add this to startup function for packer

   ```
   use { 'ThePrimeagen/git-worktree.nvim' }
   ```

2. Source the packer.lua file and run the below command

   ```
   :PackerInstall
   ```

3. Setup keybinds

   ```
   -- telescope list git_worktree
   vim.keymap.set(
       "n",
       "<leader>wt",
       ":lua require('telescope').extensions.git_worktree.git_worktrees()<cr>"
   )

   -- telescope create git_worktree
   vim.keymap.set(
       "n",
       "<leader>cwt",
       ":lua require('telescope').extensions.git_worktree.create_git_worktree()<cr>"
   )
   ```

neovim keybinds

1. Telescope list worktrees: `<leader>wt`
2. Telescope create worktree: `<leader>cwt`

2. Git Submodules

Git submodules allow you to keep a git repository as a subdirectory of another git repository. Git submodules are simply a reference to another repository at a particular snapshot in time. Git submodules enable a Git repository to incorporate and track version history of external code.

add

Used to add a new submodule to a existing repository.

```
git submodule add <upstream>
```

```
git-submodule-demo (main) git submodule add git@github.com:prajwal-badiger/dsa.git
Cloning into '/home/prajval/workspace/notes/git/git-submodule-demo/dsa'...
remote: Enumerating objects: 36, done.
remote: Counting objects: 100% (36/36), done.
remote: Compressing objects: 100% (19/19), done.
remote: Total 36 (delta 7), reused 35 (delta 6), pack-reused 0
Receiving objects: 100% (36/36), 4.15 KiB | 4.15 MiB/s, done.
Resolving deltas: 100% (7/7), done.
git-submodule-demo (main)
```

If we check git status, there are now two new files in the repository .gitmodules and dsa.

```
git-submodule-demo (main) git status -s
A  .gitmodules
A  dsa
git-submodule-demo (main)
```

.gitmodules file contains the new submodule mapping.

```
git-submodule-demo (main) cat .gitmodules
[submodule "dsa"]
    path = dsa
    url = git@github.com:prajwal-badiger/dsa.git
```

init

initialize your local configuration file.

```
git submodule init
```

update

update is run after init to fetch all the data from the project and to checkout appropriate commit listed in the subproject.

```
git submodule update
```

git clone --recurse-submodules

If you pass --recurse-submodules to the git clone command, it will automatically initialize and update each submodule in the repository, including nested submodules if any of the submodules in the repository have submodules themselves.

```
git clone --recurse-submodules <upstream>
```

If you already cloned the project and forgot `--recurse-submodules`, you can combine the `git submodule init` and `git submodule update` steps by running `git submodule update --init`. To also initialize, fetch and checkout any nested submodules, you can use the foolproof `git submodule update --init --recursive`.

3. Git usefull commands

3.1 git clone with branch

```
git clone -b <branch-name> --single-branch <upstream>
```

3.2 create and checkout to new branch

```
git checkout -b <branch-name>
```

4. Miscellaneous

4.1 Adding images to README.md

- go to the github page of the repository and click on issues
- click on new issue
- click on “Attach files by dragging & dropping, selecting or pasting them.”
- select the image you want to upload
- now copy and paste the generate output in README.md file

References

1. worktree
   https://git-scm.com/docs/git-worktree
   ThePrimeagen/git-worktree.nvim
   Git's Best And Most Unknown Feature
2. submodules
   https://git-scm.com/book/en/v2/Git-Tools-Submodules