RISC-V

System-on-Chip Design

Harris, Stine, Thompson, & Harris

Appendix C: Version Control using Git

Appendix C :: Topics

Intro to Version Control & Git

- **Version control:** Method for keeping track of changes to a set of files
  - **Example:**
    - paper_old.docx → paper_final.docx → paper_final_ah.docx → paper_final_notkiddingthistime.docx
    - These can be confusing (which was the older version?) and you may overwrite each other's work
  - Instead, use Git.

Git more capable than what's covered here. Learn more about Git from the freely available *Pro Git*: https://git-scm.com/book/en/v2

Git was developed by Linus Torvalds in 2005. (And, yes, he was also the main developer of Linux.)

Repository and Versions

- **Git generally functions like this:**
  - Maintains a **repository** (also called simply “repo”) of all files in a project.
  - Each user **clones** entire repository to own *working copy*.
  - Users can **modify** repository and test changes locally.
  - Then users can **push** changes to the main repository to create a new snapshot.
  - Git maintains a **snapshot** of repository each time user checks in files.
  - Git can revert to a previous snapshot if needed.

Do not place executables in a Git repository – instead, include Makefile and sourcefiles

Appendix C: Git

Gitting Started

Set Up Username & Email

- **After logging into your server, set up username & email:**

```
$ git config --global user.name ″Ben Bitdiddle″
$ git config --global user.email ″ben_bitdiddle@wally.edu″
```

Note: the username need not be the same as what you use at github.com. But the email must be the same.

- **Configure Git to merge:**

```
$ git config --global pull.rebase false
```

Set Up GitHub Account

- To use GitHub, sign up for an account at: **github.com**
- Set up key-based authentication (instead of a password):
  - SSH key pair: public & private key
  - Private key remains on client (never share this!)
  - If it's first time using ssh, generate a public-private key pair:

```
$ ssh-keygen -t ed25519 -C "ben_bitdiddle@wally.edu"
```

  - Use default file location and add passphrase (if desired)

Set Up GitHub Account, cont'd

- **Add key to ssh-agent by:**
  - **Creating ~/.ssh/config** file with contents:

```
Host *
  AddKeysToAgent yes
  UseKeychain yes
  IdentityFile ~/.ssh/id_ed25519
```

  - **Adding SSH key**:

```
$ ssh-agent /bin/sh
$ ssh-add -k ~/.ssh/id_ed25519
$ exit
```

  - **Printing public key** to screen:

```
$ cat ~/.ssh/id_ed25519.pub
ssh-ed25519
AAAAC3NzaC1lZDP1NTE5AAAAIMpSWKpfQ7DI3l3yywr8VdXvE+Jj/RS
xngyKgIc4MwVL
ben_bitdiddle@wally.edu
```

Set Up GitHub Account, cont’d

- **Add key to ssh-agent by (cont’d):**
  - **Uploading your public key to remote host**:
    - At **github.com**, click on **user icon** (upper right of webpage)
    - Select **Settings**
    - Click on **SSH and GPG** tab
    - Select **New SSH Key**, then copy/paste public key (both lines) into Github key field:

```
ssh-ed25519 AAAAC3NzaC1lZDP1NTE5AAAAIMpSWKpfQ7DI3l3yywr8VdXvE+Jj/RSxngyKgIc4MwVL
ben_bitdiddle@wally.edu
```

    - Set **title** to name of your server (tera.eng.hmc.edu)
    - Click **Add SSH key**

If you encounter issues, see GitHub documentation for SSH key-based authentication: docs.github.com/en/authentication/connecting-to-github-with-ssh

Appendix C: Git

Setting Up a Repository

Start New Repository

```
$ mkdir ~/my_project
$ cd ~/my_project
$ git init
$ git remote add origin https://github.com/benbitdiddle/my_project
```

A GitHub repository can be **private** or **public**. Typically, a team starts with a private repository and adds collaborators manually. Later, the repository may be made public.

Work with Existing Repository

- **Clone** it from existing repository.
- **Example:** Create clone (working copy) of Wally repository in riscv-wally within current working directory (in this case, home directory):

```
$ cd
$ git clone --recurse-submodules https://github.com/davidharrishmc/riscv-wally
```

The `--recurse-submodules` flag recursively checks out any other GitHub repositories that are submodules within the main repo.

Directions throughout the textbook assume you cloned Wally in your home directory (~).

Appendix C: Git

Basic Git Flow

Basic Git Flow: First Steps

- **Get latest version of repo** (good idea to do before each work session):

```
$ git pull
```

- **Check** that you don’t have local modified files from previous session:

```
$ git status
```

- **Modify** files (test before pushing them back into repo).

Basic Git Flow: Modifying Files

- **Move file:**

```
$ git mv <old name> <new name>
```

- **Check history of file**:

```
$ git log or $ git diff or $ git log –follow <new name>
```

git log/git diff only show history up till last move. git log -follow… gives full history

- **After happy with changes:**
  - Check status again (modified files listed under “Changes not for staged commit”; new files under “Untracked files”):

    ```
    $ git status
    ```

  - If any file changes are bad, revert to one from repository:

    ```
    $ git checkout –– <file>
    ```

Basic Git Flow: Committing Files

- **Committing & Pushing Files to Repository:**
  - If added/deleted files (deleted files remain in prior versions):

```
$ git add <file>
```
(for added files)

```
$ git rm <file>
```
(for deleted files)

  - Check status again (all files should be listed under “Changes to be commited”):

```
$ git status
```

  - Commit (use –m to add meaningful message):

```
$ git commit –m ″Fixed undeclared mmu/PhysAdr signal causing X in simulation″
```

or adds all the tracked files and then commit at the same time:

```
$ git commit –a –m ″message″
```

  - Push to remote repository (make sure all code is tested before pushing):

```
$ git push
```

Basic Git Flow: Check Commits

- **Check log of commits** (a commit is identified by a hash (unique 40-digit hexadecimal number):

```
$ git log
commit 14d3059433e212205ebf30b64ffe71d467dabb94
Author: David Harris <david_harris@hmc.edu>
Date:   Fri Jan 21 00:12:14 2022 +0000

    Fixed path to riscvOVPsimPlus

commit 55e4d09084caa95cebfc36b778de16f5b8e051b3
Author: Ross Thompson <ross1728@gmail.com>
Date:   Thu Jan 20 16:39:54 2022 -0600

    Factored out InstrValidNotFlushedM from each csr*.sv
to csr.sv
...
```

Appendix C: Git

Merge Conflicts

Resolving Merge Conflicts

- Suppose Ben and Alyssa both modified hello.c
  - After Alyssa pushes changes and then Ben pulls, **Git reports conflict:**

```
CONFLICT (content):  Merge conflict in hello.c
Automatic merge failed; fix conflicts and then commit the result.
```

  - Ben **opens conflicting file**, hello.c, which shows conflict between Ben's HEAD and Alyssa's …925 snapshot:

```
<<<<<<< HEAD
printf("Hello WORLD!");
=======
printf("Hello World!!!!!");
>>>>>>> 0b2d6c97f4265819281bf1a8d77698eb9ff30925
```

  - To fix the merge conflict, Ben must delete these 5 lines and replace them with desired code, for example:

```
printf("Hello Alyssa!!!!!");
```

  - Then Ben pushes to remote repo.

Appendix C: Git

Branching & Merging

Branching

- By default, Git keeps your work in the **main branch**
- When multiple groups working on different parts of repo, often useful to **branch repo** and then merge back together once the branch is stable.
- **Example:**
  - Create branch called muldiv:
    ```
    $ git branch muldiv
    ```
  - Move HEAD (points to working branch) to point to muldiv – HEAD points to main branch until it is changed:
    ```
    $ git checkout muldiv
    ```
- The first time pushing to a new branch, must also set origin:
  ```
  $ git commit –a –m “New muldiv branch changes”
  $ git push –u origin muldiv
  ```

Switching Branches & Merging

- While working on a branch, if you want to return to work on the main branch:

```
$ git checkout main
```

- Now, to merge changes from muldiv branch back into main branch:

```
$ git merge muldiv
```

Other Branch-Related Commands

- List branches that exist:

```
$ git branch
```

- Delete muldiv branch after it is merged into main:

```
$ git branch –d muldiv
```

- If commits in branch have not been merged, but want to force delete, use -D.

```
$ git push –d origin <branch name>
```

Appendix C: Git

Tags

Tags

- **Tags:** Name snapshots for easy (human-readable) access in the future.
- **Example:** name current snapshot as v2.0:

```
$ git tag –a v2.0 –m ″Version 2.0 Released by Ben
Bitdiddle 30 November 2021″
```

- **List** available tags:

```
$ git tag
v0.1
v1.0
v2.0
```

- By default, tags are local to user. To **push tag to repository**:

```
$ git push v2.0
```

- **Check out a tagged snapshot:**

```
$ git checkout v2.0
```

Appendix C: Git

Staging & Undo

File States

- Each file in git repository is in one of **three states**:
  - **Committed:** when repo first cloned (no changes made)
  - **Modified:** when changed files or metadata (permissions, etc.)
  - **Staged:** `git add` moves file's state from modified to stage

Undoing Changes

- **Undo file changes** (revert to main branch version of file):
  - State goes from **modified** to **committed**.
  - Local changes are **discarded.**

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

- **Unstage file:**
  - State goes from **staged** to **modified**.
  - Local changes are **not discarded.**

```
$ git reset <file>
```

- **Uncommit file:**
  - State goes from **committed** to **modified**.
  - Local changes are **not discarded.**
  - Completed for entire repo (otherwise use git checkout <file>)

```
$ git reset --hard hash#
```

Revert to Previous Version of Repo

- To roll back a recent change that you pushed:

```
$ git revert <hash>
```

  - `<hash>` is the snapshot to undo

IMPORTANT: Never perform `git reset` if commits are already pushed. Rewriting published history creates problems for other users.

Appendix C: Git

Submodules

Submodules: Other Repositories

- Repositories often use code from other repositories.
- For example, Wally uses test cases from the riscv-arch-test repository.
  - If simply copied code from that repository into riscv-wally, it would soon be out of date.
  - Instead, Git supports using other Git repositories (called submodules) within another repository

Submodules are usually owned by somebody else, and you should not modify it or attempt to push commits back into it.

Submodules: Other Repositories

Incorporate one repository within another:

```
$ git submodule add <URL>
```

Update to the latest version of a submodule:

```
$ git submodule update --remote
```

Pull your main repository and automatically update any submodules

- When submodules are added to a repository, git pull does not fetch the new submodules. Force the fetch using `--recurse-submodules`):

```
$ git pull --recurse-submodules
```

To clone a repository with submodules:

- Be sure to include the `--recurse-submodules` flag so you don't have to initialize and update each submodule manually.

```
$ git clone --recurse-submodules <URL>
```

Appendix C: Git

Other Git Capabilities

Moving Between Branches

- Temporarily save (stash) changes in current branch (so can change to other branch but without discarding existing work):

```
$ git stash
```

- Then change to other branch & do work. When done, restore work in progress:

```
$ git stash apply
```

Ignoring Files in Repository

- Don’t want to put all files in repo (i.e., executables, object files, etc.).
  - Add them to .gitignore file
  - .gitignore usually in repo’s root directory (but subdirectories may also have their own .gitignore)
  - Example .gitignore

```
*.o
*.objdump
examples/C/sum/sum
examples/C/fir/fir
```

Compare Snapshots

- By default, compares snapshot with HEAD:

```
$ git diff c30
```

About these Notes