Step-by-step “git”

Michael Nahas

Why do we need git?

Many people editing the same files.

We want:

- to not lose anyone's changes
- to have a book that compiles

Git is a tool for storing and exchanging changes.

Git has features the reviewers want.

If you follow this guide, we won't lose anything.

How we use “git” and “GitHub”

- The shared copy of the book files is on github.com
- You will “clone” GitHub's repository to make your own local copy of the book.
- After making changes, you will “commit” them locally.
- After a commit, you can “pull” the latest version of the book from GitHub (with your changes).
- When you are ready for others to see your changes, you will “commit”, then “pull” (to include everyone's changes) and then “push” to GitHub.

Install git

Will vary by OS.

- http:/git-scm.com/downloads

Remember to set user.name and user.email

- `git config --global user.name "James Bond"`
- `git config --global user.email "007@mi6.gov.uk"`

GitHub Account

To share your edits, you need a GitHub account.

- http://github.com

Sign up and email your username to Mike S.:

- Michael Shulman viritrilbia@gmail.com

“clone” the GitHub repository

The commands are:

- `cd` *`dir_where_you_want_book_dir`*
- `git clone https://github.com/HoTT/book.git`
- `cd book`
- …

After “clone”, all commands must be run in “book/”!

“commit”ing your changes.

After editing files, you can run:

- `git add --all`
- `git commit -m` *`“description of edit”`*

- *“commit” stores your version of the files in git.*
- *An expert can undo/redo any committed change.*
- *This command will record all* files in the directory!*
- *Commit every time you change tasks/subjects.*

Commit often.

“pull” in other people's changes

After committing, the command is:

- `git pull`

This command fetch's other people's changes and merges them with your own.

*If you see “merge conflict” or “unable to merge” or “unresolved changes”, **get an expert!***

“push”ing your changes to everyone

After committing AND pulling, the commands are:

- `latex main.tex`
- `git push`
- *enter GitHub username + password*

This command copies your version to GitHub. Your version will become the shared version.

If you see “not a fast forward...”, that means you need to rerun “git pull” and try again to compile and “git push”.

“forked” GitHub projects

GitHub has a good description of clone + config:

- https://help.github.com/articles/fork-a-repo

The only change is that the “pull” command is:

- `git pull upstream`

You still need to “commit” before “pull”ing.

Setup Summary

- Install git
  - **set `user.name` and `user.email`**
- Get GitHub account
  - **email your username to Andrej**
- “clone” the GitHub repository

Usage Summary

- Record your changes in git
  - `git add --all && git commit -m “...”`
- Get other people's changes from GitHub
  - `git pull` (*or* `git pull upstream`)
- Send your changes to GitHub
  - `latex main.tex && git push`

Who is an expert?

Experts:

- Mike Nahas mike@nahas.com
- Andrej Bauer andrej.bauer@andrej.com

Experienced git users:

- Michael Shulman viritrilbia@gmail.com
- Peter Lumsdaine p.l.lumsdaine@gmail.com

Additional commands

- Which files did I change since my last commit?
  - `git status`
- What changes did I make in those files?
  - `git diff`
- Who wrote the crap in this file?
  - `git blame`

Hidden files

- `.git/`
  - Stores all the data, config, etc.
  - If you delete it, you lose everything.
- `.gitignore`
  - File that tells git not to check-in *.pdf, *.dvi, etc.

End of Talk

Additional Slides

- Anything that follows is probably incomplete...

Why Source Control? and How?

Michael Nahas

Why Source Control?

Have you ever …

- accidentally deleted a source file?
- kept alternate versions of a source file?
  - in “foo.ml.bak”, “foo.ml.bak” …
  - or in “old_version/” “old_old_version/”...
- broken the build and wanted to know what you changed to hose yourself?
- edited the same file on two different computers?

What is Source Control?

- “Source control” is a process for managing multiple copies of a project over time.
- You've been doing it already.
- Good news! There are tools to help you!

While “source control” is perceived as “being for groups”, **none** of the problems on the previous slide involve another person.

What does git do?

Git creates and manipulates “snapshots” of a directory (and its subdirectories).

It is most often used for source control, but is a general tool with many uses.

What can git do?

- Create a new snapshot (“git commit”)
- Compare two snapshots (“git diff”)
- Receive snapshots (“git fetch”, “git pull”)
- Send snapshots (“git push”)
- Restore files of snapshot (“git checkout”)
- Name a snapshot (“git tag”)

Git organizes snapshots.

Snapshots are organizes as “commit”s.

A commit contains:

- The snapshot
- A description containing:
  - The author
  - The date-time
  - A short and long description
  - The parent “commit”(s).

What *more* can git do?

- Look at the ancestors of a commit
  (“git log”, “gitk”)
- Merge snapshots with a common ancestor*
  (“git merge”, “git rebase”, “git cherry-pick”)
- Find when changes entered a file (“git blame”)
- Name a version being worked on (“git branch”)
- Revert to a previous version (“git reset”)

Git without “NEXT”

Git has a feature know by various names

- “stage” or “staging area”
- “the index”
- “the cache”

It is very powerful and it is required if you ever want to split one edit into two sequential edits.

For simplicity, I will **NOT** be covering this feature.

Creating a new repository.

Git stores all its data in a “repository”.

This includes snapshots, config, logs, etc.

To create a new repository:

- `cd` *`dir_name`*
- `git init`

If files already exist, you want to:

- Edit “.gitignore”
- `git add .`
- `git commit`

Copying an existing repository

- `git clone` *`URL`*

For example,

- `git clone` *`https://github.com/HoTT/book.git`*

What did I just do?

- Created a directory “.git” to hold the repository
- Created a branch called “master”.
- “checked out” all the files in master.
- Set “HEAD”, the current branch, to “master”.

If you cloned a repository, you also:

- Created a default remote repository, “origin”.
- Copied the complete history of the “master” branch from “origin” as branch “origin/master”

Identify yourself to git.

```
git config --global user.name "Michael Nahas"
git config --global user.email "mike@nahas.edu"

git config --global core.editor emacs
```

.gitignore

You don't want git to take a snapshot of all files.

- Editors create tmp files and backup files
- Compilers generate output files
- OSX creates cache directories “.DS_Store”

You can specify files and patterns in “.gitignore”

WARNING: Git does not handle empty directories.
The workaround is to put a .gitignore file in it.

See what you will commit.

- `git status`
  - Tells you what branch you're editing
  - Tells you what files have changes
  - Tells you what commands will restore files

Example output:

```
# On branch ssreflect
# Untracked files:
#   (use "git add <file>..." to include in what
will be committed)
#
#  ssrplugin/Makefile.coq
```

Creating a new commit

- `git add --all`
- `git commit -m` *`“message”`*

This takes two commands, so that we avoid the “NEXT” issue.

“git commit” will start $EDITOR or “vi”...

- The first line of the file is the short description
- The rest of the file is the long description

See your changes

- `git diff HEAD~1`
- `git diff origin/master`
- `git diff HEAD~3 – foo.txt`
- `git diff 0e444a8d`

Branches

Create a new branch

- `git checkout -b` *`branch_name`*

“git branch” exists, but doesn't change HEAD!

Change to a branch

- `git checkout` *`branch_name`*

Current branch moves with a commit.

Tags do not. (So use tags for version ids, etc.)

Receiving data

- `git fetch origin`

If you want to add a new remote respository:

- `git remote add` *`name URL`*

- Only commits – no snapshots
- Say what it can do – no commands
- Picture + usage on github
- Only branches are master and origin/master
- Only book example – some files in one directory
- Clone, add+commit, pull, push