Learning by Doing
A Short Introduction to git

Jianwen Wei
weijianwen@sjtu.edu.cn

Omni-Lab, Shanghai Jiaotong University
http://omnilab.sjtu.edu.cn

Aug 21, 2013

What is git?

What does a Version Control System do?

- Track source code
  - Maintain code history, integrity, atomic change...
- Coordinate distributed development
  - branch, merge conflicts, tag...

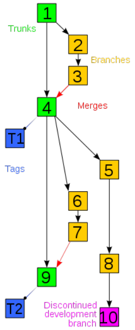

Figure: VCS work flow

VCS Work Flow Categories

- Centralized: VSS, CVS, SVN
- Distributed: BitKeeper, git, mercurial...

Distributed VCSs support centralized work flow too.

Why git is better than X (SVN, CVS, ...)

- git is super fast
- Full repository clone
- Local history: no need to connect to servers when viewing the revision history
- Cheap branch and easy merge
- Lots of git host choices: github, Google Code, gitbucket, gitlab, CodePlex...
- Other things: tidy working directory, better compression, multi work flow support, ...

General Advice on Learning git

- Try git and github
- Most graphical tool/plug-ins[1] *suck*. Please use the command-line git.
- Read git's prompts, run **git help** to get help.
- Find "how-to" on Google, StackOverflow, git book.

[1]tortoisegit, gitk, EGit, Snow Octocat... But please, oh please use the command-line tool.

Rules of Thumb for git

- “A clear development flow is worth thousands of VCSs.”
- One repo for one project. Use `submodule` to organize super projects.
- Modular design, avoid simultaneous source file editing by different members.
- Head version at trunk is always ready to deploy.
- Modification is made on branches, then merged into trunk.
- Stay on your own branch.
- Write comment to each commit.

git work flow

git's stand-alone work flow

You can use git on a stand-alone computer and easily integrate the code into a more sophisticated work flow (distributed or centralized) at a later time.

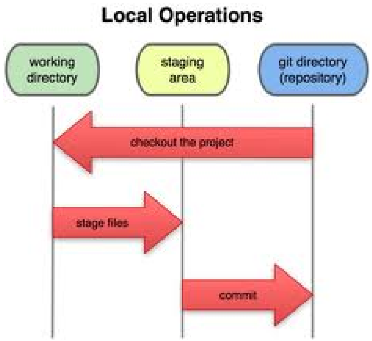

Figure: git's local work flow

git's distributed work flow

- Every collaborator keeps a full clone of the repository.
- All repositories are peers.
- Repositories are not necessarily consistent at all time. Use push/pull to exchange changes when necessary.

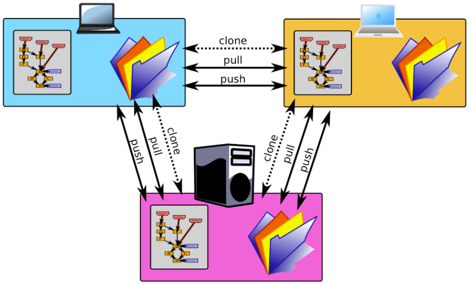

Figure: git's distributed work flow

git's emulation to the centralized work flow (**RECOMMENDED**)

- Pick a git repo as the central one with which other repos sync
- The statement, "all repositories are peers.", still holds.
- We pretend that we see the central repo only, unaware of each other's peer repo.

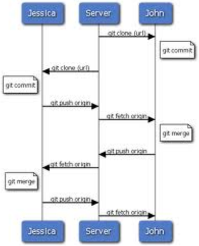

Figure: git's centralized work flow for John and Jessica

Set up git

- Please follow github's nice tutorials to set up git on Windows, Linux or Mac.
- Must-known things about SSH keys: private key, public key, the pass phrase to access the private key, key fingerprint.
- Don't forget to set `user.name` and `user.email` before your very first git command-line commit.

git command

The most useful git command

- `help`
- `init`
- `status`
- `add`
- `commit`
- `diff`
- `tag`
- Working with `branch`
- Working with `remote`
- `submodule`
- Oh, there is a conflict!!!
- Time Machine

`help`: Get help

```
git help add
git help commit
git help rebase
```

`init`: Initialize a local git repo for your project

```
cd YOUR_PROJ_DIR
git init .
```

status: Show the status of your repo

git status also tells you how to *undo* the last operation on git.

```
git status
```

File status in git: untracked, unstaged, staged (indexed), committed.

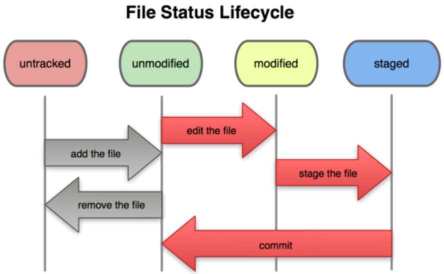

Figure: File status life cycle

add: A multi-function git command

```
git add FILES_OR_DIR_LIST
```

- For untracked files: add them to git's control.
- For unstaged changes: add them to the staged area.
- For conflicted files: `add` marks them as "resolved".

.gitignore: Ignore files

- Ignore all *.tmp, *.old and files in dir tmp/.

```
.tmp
.bak
targets/*
```

- Do *NOT* ignore file results.txt in dir targets/.

```
!targets/results.txt
```

- For more information, try `git help gitignore`.

commit: Store the status (snapshot) permanently

- `git commit -m "YOUR_COMMENT"`
  - Stores the STAGED changes only
- `git commit -a -m "YOUR_COMMENT"`
  - Stores all the STAGED and UNSTAGED changes.
- Each commit is identified by a **UNIQUE** SHA-1 ID of 40 ASCII characters.

```
commit dd5f924c40096b9cda27ffd1cfd1205822ab3c70
Author: Github Support <me@github.com>
Date:   Sun Apr 1 19:38:37 2012 +0800

   Restart the git-tutorial project.
```

diff: Find differences

- `git diff`
  - changes between the staged and working files
- `git diff --staged`
  - changes between the HEAD and the staged files
- `git diff HEAD`
  - changes between the HEAD and the working files
- `git diff COMMIT_ID COMMIT_ID`
  - changes between two commits

tag: Mark a milestone version

- `git tag`
  - See all the tag
- `git show TAG_NAME`
  - See a tag in detail
- `git tag TAG_NAME`
  - Add a “lightweight” tag
- `git tag -a TAG_NAME -m YOUR_COMMENT`
  - Add an anotated tag
- `git tag -d TAG_NAME`
  - Delete a tag

Submodule: Integrate multi git repos

```
git submodule add REPO_URL LOCAL_PATH
```

- Repo in Repo
- Manage other repos as “submodules” in your project

Working with **branch**: branch, checkout, merge

A branch-based development flow:

1. Create a branch
2. Switch to the newly-created branch
3. Modify and commit on the branch
4. Merge branch's changes into trunk.

Working with **branch**: `branch`, checkout, merge

- `git branch`
  - See all the branches
- `git branch BRANCH_NAME`
  - Create a branch
- `git branch -d BRANCH_NAME`
  - Delete a branch
- `git branch -D BRANCH_NAME`
  - Force delete a branch

Working with **branch**: branch, `checkout`, merge

- `git checkout BRANCH_NAME`
  - Switch to a branch. The working files will change.
- `git checkout -f BRANCH_NAME`
  - Force switch to a branch
- `git checkout master`
  - Go back to trunk, named *master* in git.
- `git checkout -b BRANCH_NAME`
  - Create a branch then switch to it.

Working with **branch**: branch, checkout, `merge`

- `git merge BRANCH_A BRANCH_B`
  - Merge branch_a's and branch_b's changes into *current* branch
- `git checkout master, git merge master BRANCH_NAME`
  - Merge changes into trunk, the master branch.

Working with **remote**: `clone`, remote, push, pull

- `git clone REPO_URL` Full clone of a repo.
- URL can be in forms of local dir (~/proj), git (git://xxx), SSH (ssh://xxx), https (http://xxx)...

Working with **remote**: clone, `remote`, push, pull

- `git remote`
  - Show all the tracked repositories.
- `git remote show REPO_NAME`
  - Show the repo's details.
- `git remote add REPO_NAME REPO_URL`
  - Add a remote repo to tracked list.
- `git remote rm REPO_NAME`
  - Remove a remote repo from the tracked list.
- `git remote rename REPO_OLD REPO_NEW`
  - Rename a repo.
- `git help remote`
  - Show remote help doc

Working with **remotes**: clone, remote, `push`, `pull`

- `git pull REPO_NAME REMO_BRANCH`
  - Merge remote branch's changes into current branch.
- `git push REPO_NAME REMO_BRANCH`
  - Push current branch's changes to the remote branch.
- `git push REPO_NAME :REMO_BRANCH`
  - Delete a remote branch.

Oh, there is a conflict!!!

- A conflict looks like:

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

- Conflicts arise when git cannot automatically merge changes at `merge` or `pull` operations.
- Don't panic. Conflicts are no big deal, sometimes even inevitable.
- What you should do: merge the conflicts, mark the files as "resolved", then commit the changes.

Working with conflicts: merge, resolve, commit

1. You can edit the conflicted files, merge conflicts MANUALLY. Or,
   - `git checkout --theirs FILES` replace the conflicted files with *theirs*.
   - `git checkout --ours FILES` replace the conflicted files with *ours*.
2. `git add CONFLICT_FILES` mark the file as resolved.
3. `git commit -m "YOUR_COMM"` commit changes to the repo.

"Time Machine": `stash`, checkout

`stash` saves your temporary work and resets the files to HEAD version. You can handle some emergency fix first then continue to hack at a latter time.

1. Save the temp changes.

```
git stash
```

2. Check the stash list.

```
git stash list
```

3. EDIT and COMMIT your emergency fix.
4. Continue to hack.

```
git stash pop
```

"Time Machine": stash, `checkout`

`checkout` enable you to go backward and forward in the revision history.

1. `git checkout COMMITID_OR_TAGNAME`
    - Time Machine starts up.
2. You are on a `unnamed` branch with file status dating back. Do anything you want.
3. `git checkout master`
    - Come back to master.

"Time Machine": stash, `checkout` (continue)

`git checkout COMMIT_ID -- FILE_LIST` check out the file list at the specified commit.

Exercises

Exercise: Set up git environment

1. Set up git on your computer, and sign up a github account.
2. Initialize a local project as git repo, make your first git.

Exercise: git basics

Be familiar with `status`, `add`, `commit`, `diff`, `tag`.

Exercise: Branch-based development

1. Create a branch.
2. Checkout to that branch.
3. Merge the changes into trunk (master).
4. Delete the branch.

Exercise: Handle conflicts

1. Create a local branch called `brA`, modify a text file.
2. Create a local branch called `brB`, modify the text file on the same line as brA.
3. Merge `brA` into master, then merge `brB` into master. So a conflict arises.
4. Resolve the conflict, then add, commit.

Exercise: Time Machine

Use `stash`, `checkout` to do time travle.

Exercise: Fork — Be social on github

1. Register a github account and leave your email address public on your homepage.
2. Open an issue in GitForBeginners to say hello.
3. Fork GitForBeginners.
4. Now go to your github homepage, you will find a clone of GitForBeginnerss there.

Exercise 5: Manage remotes

1. Clone *your* GitForBeginners.
2. Show the remote repo aliases with:

```
git remote -v
```

3. Rename remote alias `origin` to a *name you like* with git remote rename origin NAME_YOU_LIKE
4. Add GitForBeginners as the upstream repo with

```
git remote add upstream \
git@github.com:weijianwen/GitForBeginners
```

Exercise: Remote branch on github

1. Create a local branch with your full name, such as `branch-zhangsan`.
2. Switch to that branch, write something into README.mkd. Then push this branch to your github repo.

The final challenge: Send a merge request

Send me a merge request on github. That is, ask me to merge from your YOUR_NAME branch in your GitForBeginners repo, into the master branch in my GitForBeginners repo.

Congratulations! You will get your gitlab account after this challenge. Please check your mailbox.

Appendix

Reference and more information

- "Git Tutorials" by Li Yanrui
- github:help
- Pro Git On line
- Video: "Git the basics" by Bart Trojanowski
- O'Reilly Book: Version Control With Git, 2nd Edition
- "Git Reference"
- "GitforBeginners" by Jianwen Wei, hosted on github

Acknowledgement

- The slides are composed with Markdown language, and converted to latex beamer with pandoc.
- XeTeX is a nice typesetting system. latexmk helps to hide the complexity of compilation.
- The slides, along with the project, is hosted on github.
- Feedback is always welcomed. Write me or open an issue on the project homepage.