Git Internal and Working with Git

Outline

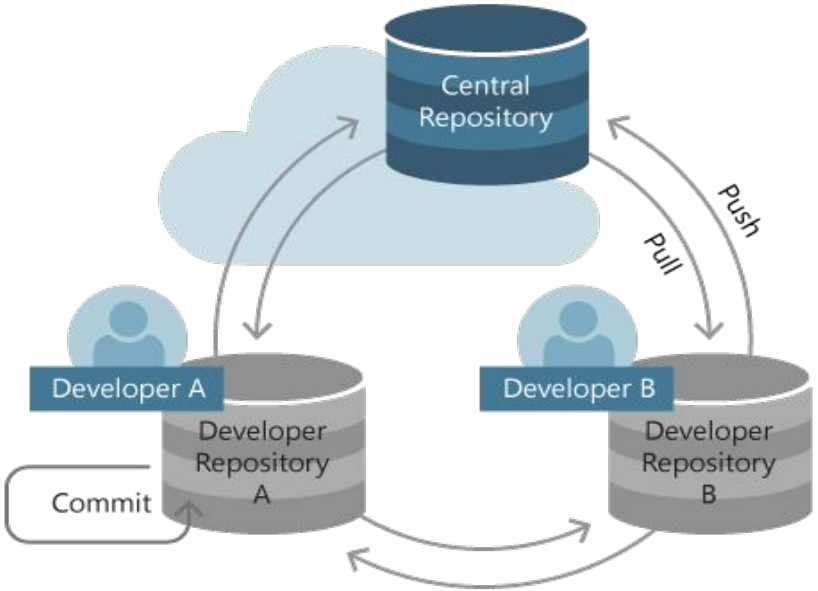
1. Basic workflow
2. Git repository
3. Create a new repository
4. Clone an existing repository
5. Recording changes
6. Viewing the commit history
7. Undoing Things
8. Working with Remotes
9. Tagging
10. Git Aliases

1. Basic workflow

Your local repository consists of three "trees" maintained by git. the first one is your Working Directory which holds the actual files. the second one is the Index which acts as a staging area and finally the HEAD which points to the last commit you've made.

2. Git repository

A Git repository is a virtual storage of your project. It allows you to save versions of your code, which you can access when needed.



3. Create a new repository

Initializing a Repository in an Existing Directory

If you're starting to track an existing project in Git, you need to go to the project's directory and type:

```
$ git init
```

This creates a new subdirectory named .git that contains all of your necessary repository files – a Git repository skeleton. At this point, nothing in your project is tracked yet. (See Chapter 10 for more information about exactly what files are contained in the .git directory you just created.)

4. Clone an existing repository

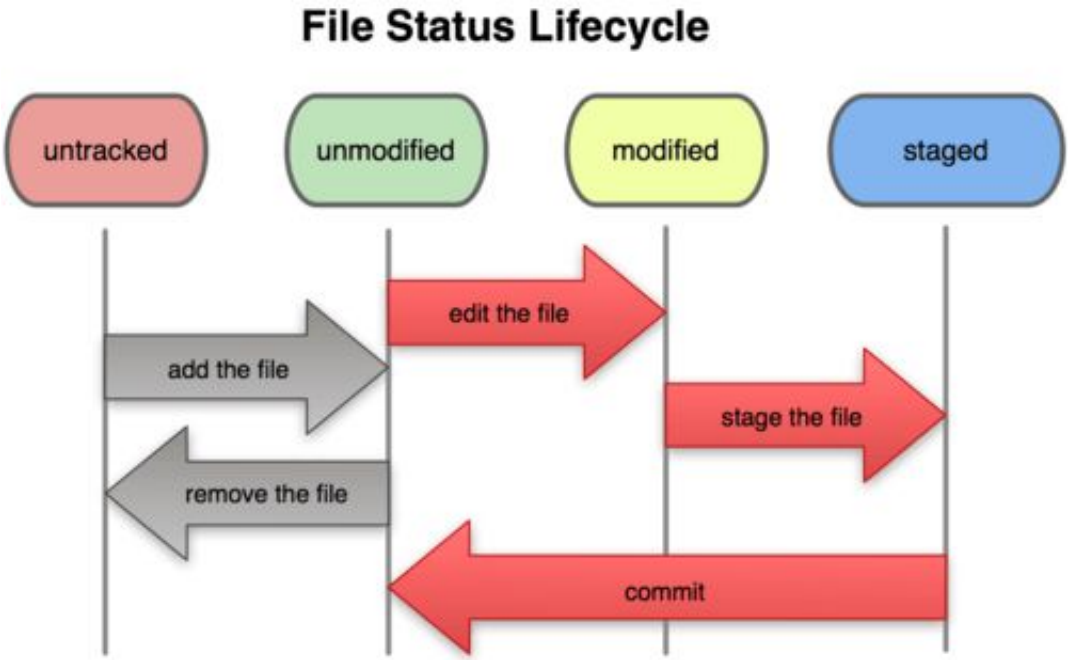
You clone a repository with git clone [url]. For example, if you want to clone the Git linkable library called libgit2, you can do so like this:

```
$ git clone https://github.com/libgit2/libgit2
```

That creates a directory named “libgit2”, initializes a .git directory inside it, pulls down all the data for that repository, and checks out a working copy of the latest version. If you go into the new libgit2 directory, you’ll see the project files in there, ready to be worked on or used. If you want to clone the repository into a directory named something other than “libgit2”, you can specify that as the next command-line option:

```
git clone https://github.com/libgit2/libgit2 mylibgit
```

5. Recording changes



5. Recording changes

Checking the Status of Your Files

The main tool you use to determine which files are in which state is the git status command. If you run this command directly after a clone, you should see something like this:

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```

```
$ echo 'My Project' > README
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    README

nothing added to commit but untracked files present (use "git add" to track)
```

5. Recording changes

Tracking New Files

In order to begin tracking a new file, you use the command git add. To begin tracking the README file, you can run this:

```
$ git add README

$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README
```

5. Recording changes

Staging Modified Files

Let’s change a file that was already tracked. If you change a previously tracked file called CONTRIBUTING.md and then run your git status command again, you get something that looks like this:

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   CONTRIBUTING.md
```

5. Recording changes

Short Status

While the git status output is pretty comprehensive, it's also quite wordy. Git also has a short status flag so you can see your changes in a more compact way. If you run git status -s or git status --short you get a far more simplified output from the command:

```
$ git status -s
 M README
MM Rakefile
A  lib/git.rb
M  lib/simplegit.rb
?? LICENSE.txt
```

5. Recording changes

Ignoring Files

Often, you'll have a class of files that you don't want Git to automatically add or even show you as being untracked. These are generally automatically generated files such as log files or files produced by your build system. In such cases, you can create a file listing patterns to match them named .gitignore. Here is an example of .gitignore file:

```
# no .a files
*.a
# but do track lib.a, even though you're ignoring .a files above
!lib.a
# only ignore the TODO file in the current directory, not subdir/TODO
/TODO
# ignore all files in the build/ directory
build/
# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt
# ignore all .pdf files in the doc/ directory and any of its subdirectories
doc/**/*.pdf
```

5. Recording changes

Committing Your Changes

The simplest way to commit is to type `git commit`

```
$ git commit
```

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Your branch is up-to-date with 'origin/master'.
# Changes to be committed:
#   new file:   README
#   modified:   CONTRIBUTING.md
~
~
~
".git/COMMIT_EDITMSG" 9L, 283C
```

5. Recording changes

Committing Your Changes

Alternatively, you can type your commit message inline with the `commit` command by specifying it after a `-m` flag, like this:

```
$ git commit -m "Story 182: Fix benchmarks for speed"
[master 463dc4f] Story 182: Fix benchmarks for speed
 2 files changed, 2 insertions(+)
 create mode 100644 README
```

Skipping the Staging Area

Adding the `-a` option to the `git commit` command makes Git automatically stage every file that is already tracked before doing the commit, letting you skip the `git add` part

```
$ git commit -a -m 'added new benchmarks'
[master 83e38c7] added new benchmarks
 1 file changed, 5 insertions(+), 0 deletions(-)
```

5. Recording changes

Removing Files

If you simply remove the file from your working directory, it shows up under the “Changed but not updated” (that is, unstaged) area of your git status output

```
$ rm PROJECTS.md
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    PROJECTS.md

no changes added to commit (use "git add" and/or "git commit -a")
```

5. Recording changes

Removing Files

Then, if you run `git rm`, it stages the file's removal

```
$ git rm PROJECTS.md
rm 'PROJECTS.md'
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    deleted:    PROJECTS.md
```

```
$ git rm --cached README
```

To do this, use the `--cached` option:

6. Viewing the Commit History

Viewing the Commit History

After you have created several commits, or if you have cloned a repository with an existing commit history, you'll probably want to look back to see what has happened.

```
$ git log
$ git log -p -2
$ git log --stat
$ git log --pretty=oneline
$ git log --pretty=format:"%h %s" --graph
$ git log --since=2.weeks
$ git log -S function_name
$ git log --pretty="%h - %s" --author=gitster --since="2008-10-01" \
   --before="2008-11-01" --no-merges -- t/
```

7. Undoing Things

If you commit and then realize you forgot to stage the changes in a file you wanted to add to this commit

Unstaging a Staged File

```
$ git commit -m 'initial commit'
$ git add forgotten_file
$ git commit --amend
```

```
$ git add *
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    README.md -> README
    modified:   CONTRIBUTING.md
```

7. Undoing Things

Unmodifying a Modified File

What if you realize that you don't want to keep your changes to the CONTRIBUTING.md file?

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   CONTRIBUTING.md
```

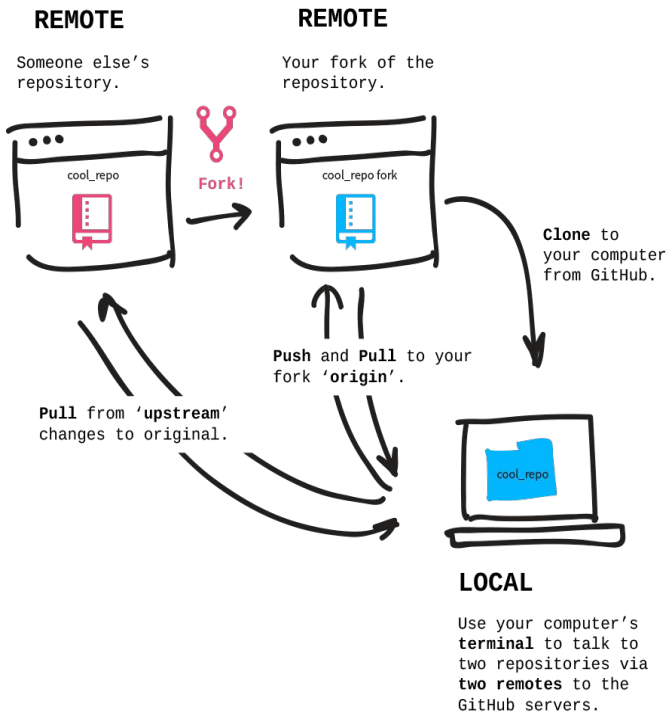
It tells you pretty explicitly how to discard the changes you've made. Let's do what it says:

```
$ git checkout -- CONTRIBUTING.md
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    README.md -> README
```

8. Working with Remotes

To be able to collaborate on any Git project, you need to know how to manage your remote repositories. Remote repositories are versions of your project that are hosted on the Internet or network somewhere. You can have several of them, each of which generally is either read-only or read/write for you.



8. Working with Remotes

Showing Your Remotes

$ git remote -v

Adding Remote Repositories

$ git remote add pb https://github.com/paulboone/ticgit

Fetching and Pulling from Your Remotes

$ git fetch [remote-name]

Pushing to Your Remotes

$ git push origin master

Inspecting a Remote

$ git remote show origin

Removing and Renaming Remotes

$ git remote rename pb paul

9. Tagging

Like most VCSs, Git has the ability to tag specific points in history as being important. Typically people use this functionality to mark release points (v1.0, and so on). In this section, you'll learn how to list the available tags, how to create new tags, and what the different types of tags are.

9. Tagging

Listing Your Tags

$ git tag

Annotated Tags

$ git tag -a v1.4 -m "my version 1.4"

Lightweight Tags

$ git tag v1.4-lw

Tagging Later

$ git tag -a v1.2 9fceb02

$ git show v1.2

Sharing Tags

$ git push origin --tags

Checking out Tags

$ git checkout -b version2 v2.0.0

10. Git Aliases

There's just one little tip that can make your Git experience simpler, easier, and more familiar: aliases
Examples you may want to set up:

You can add your own unstage alias to Git

```
$ git config --global alias.co checkout
$ git config --global alias.br branch
$ git config --global alias.ci commit
$ git config --global alias.st status
```

This makes the following two commands equivalent

```
$ git config --global alias.unstage 'reset HEAD --'
```

```
$ git unstage fileA
$ git reset HEAD -- fileA
```

Question & Answer?