Git Tutorial

Part I: Basics

Bardia Mojra
December 17, 2021
Robotic Vision Lab
University of Texas at Arlington

Outline

- Introduction
- Git Basics
- Basic operations, tips and examples
- Ignore file and what to ignore
- Read-Me file and markdown language
- Basic branching, best practices and examples
- Tagging
- Diff tools and example
- Basic merging, best practices and examples

Introduction

- Git vs. Github, Gitlab, Mercurial, SVN and etc
- Created by Linus Torvalds in 2005
- Designed for ***FAST*** for synchronization of source code across a distributed network:
  - A patch could require 250 atomic operations and all must be applied under 3 seconds, otherwise not scalable → Journaling File System (similar to Linux Kernel)
- Other design criteria [wiki/Git]:
  - Take Concurrent Versions System (CVS) as an example of what not to do; if in doubt, make the exact opposite decision.
  - Support a distributed, BitKeeper-like workflow.
  - Include very strong safeguards against corruption, either accidental or malicious.

Git Basics

1. Create or clone a repo:
   - Create repo on web then clone it to local (recommended for beginners)
   - Create repo on server, init local clone then push it
   - Clone an existing repo to local
2. Add git-ignore and read me files
3. Update git-ignore file per project needs
4. Add initial files and make initial commit before starting development

Basic Operations

1. Create or clone a repo:
   - Create repo on web then clone it to local

```
cd git
git clone https://BardiaMojra@github.com/BardiaMojra/test001.git
```

Basic Operations

1. Create or clone a repo:
   - Create repo on server, init local clone then push it

```
cd git
mkdir test001
cd test001/
echo '# test001' >> README.md
git init
git add README.md
git commit -m 'first commit'
git branch -M main
git remote add origin https://BardiaMojra@github.com/BardiaMojra/test001.git
git push -u origin main
```

Git Ignore

- What to ignore?
  - Almost everything
  - All binary files except the final output file!
  - Data
  - Dev and build files
  - Any file with unknown extensions
- What not to ignore?
  - Source code
  - Final output file (yes even if it is binary)
  - Dev meta data

Git Ignore Example

- https://github.com/github/gitignore/blob/main/TeX.gitignore

Read-Me File and Markdown Language

- Cheat sheet:
  - https://www.markdownguide.org/cheat-sheet/
- Basic Syntax:
  - https://www.markdownguide.org/basic-syntax
- Extended Syntax:
  - https://www.markdownguide.org/extended-syntax

Branching

- This is the core feature that support git agility
- Developers can create unlimited number of branches
- Developers can use arbitrary and long names for naming branches
- That’s it…
  - So the idea is that…
    - Each developer:
      - First creates their own branch to work on a specific task
      - Keep developing in an isolated branch while being able to see and compare source code in other branches
      - Once the feature is ready in a separate branch, only then it should be queued for merging in coordination with a ***maintainer*****!**

Branching Basics

https://zeroturnaround.com/rebellabs/git-commands-and-best-practices-cheat-sheet/

Create a Repository

From scratch -- Create a new local repository

```
$ git init [project name]
```

Download from an existing repository

```
$ git clone my_url
```

Observe a Repository

List new or modified files not yet committed

```
$ git status
```

Show the changes to files not yet staged

```
$ git diff
```

Show the changes to staged files

```
$ git diff --cached
```

Show all staged and unstaged file changes

```
$ git diff HEAD
```

Show the changes between two commit ids

```
$ git diff commit1 commit2
```

List the change dates and authors for a file

```
$ git blame [file]
```

Show the file changes for a commit id and/or file

```
$ git show [commit]:[file]
```

Show full change history

```
$ git log
```

Show change history for file/directory including diffs

```
$ git log -p [file/directory]
```

Working With Branches

List all local branches

```
$ git branch
```

List all branches, local and remote

```
$ git branch -av
```

Switch to a branch, my_branch, and update working directory

```
$ git checkout my_branch
```

Create a new branch called new_branch

```
$ git branch new_branch
```

Delete the branch called my_branch

```
$ git branch -d my_branch
```

Merge branch_a into branch_b

```
$ git checkout branch_b
$ git merge branch_a
```

Tag the current commit

```
$ git tag my_tag
```

Make a Change

Stages the file, ready for commit

```
$ git add [file]
```

Stage all changed files, ready for commit

```
$ git add .
```

Commit all staged files to versioned history

```
$ git commit -m "commit message"
```

Commit all your tracked files to versioned history

```
$ git commit -am "commit message"
```

Unstages file, keeping the file changes

```
$ git reset [file]
```

Revert everything to the last commit

```
$ git reset --hard
```

Synchronize

Get the latest changes from origin (no merge)

```
$ git fetch
```

Fetch the latest changes from origin and merge

```
$ git pull
```

Fetch the latest changes from origin and rebase

```
$ git pull --rebase
```

Push local changes to the origin

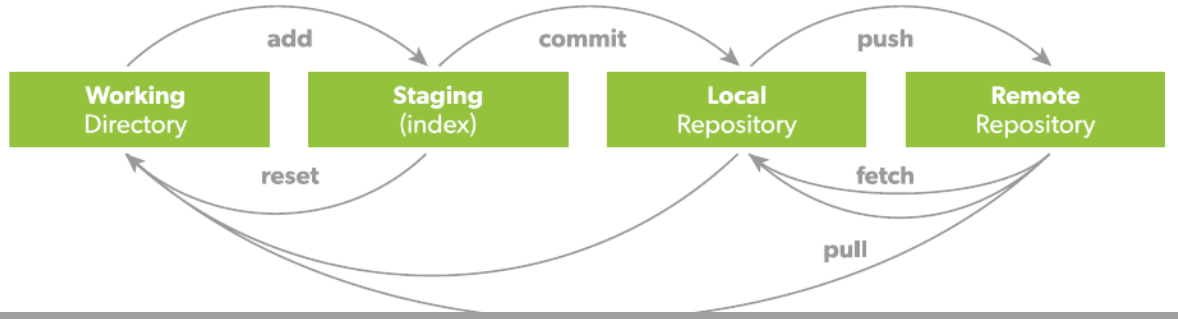
```
$ git push
```

Finally!

When in doubt, use git help

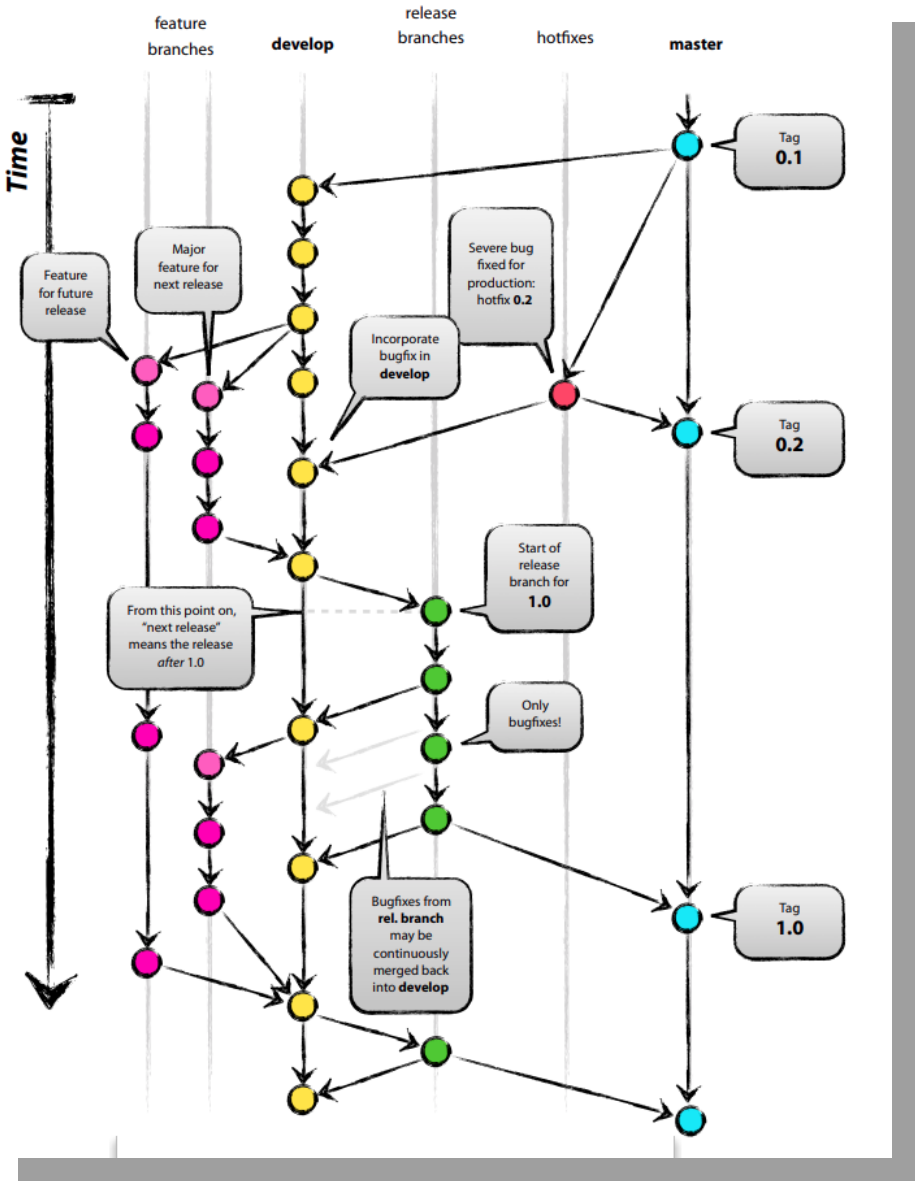
```
$ git [command] --help
```

Or visit training.github.com for official GitHub training.



Branching Strategies

- Essentially a hierarchical naming strategy
  - e.g. our paper repository branches would be something like:
    - Main (master) – unique
    - Journal_Papers_2021_Archive – unique
      - Use for backups or manually commit every 3-6 months
      - Performs no merge operation, simply archiving
    - Journal_Papers_2021_Active – unique
      - Use for active merging of paper/project directories
      - Adds extra layer of protection for active merging operation
    - JP_2021_QEKF_dev_00x – many
    - JP_2021_QEKF_update_00x – very few
      - For source code, use hotfix
      - I use dev branch instead
    - JP_2021_QEKF_merge_00x – few
    - JP_2021_QEKF_review_00x – few
    - JP_2021_QEKF_release_00x – very few
    -
- https://nvie.com/posts/a-successful-git-branching-model/



› Thank you!