GIT 101

how to git?

Summary

- History
- Why do we git
- Git basics
  - the three states
- Git CLI
  - basic snapshotting
- Branching
- Merge or Rebase ?
- Stash \o/
- checkout
- Cherry-picking
- git bisect 🙀
- Handling remote
- Git cheatsheet
- Best practices ?
- Git flow

History

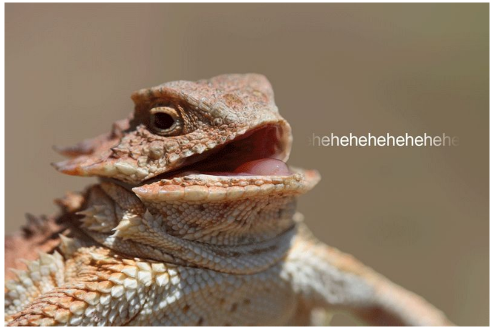
for the Linux kernel

Some of the goals of the new system were as follows:

- Speed
- Simple design
- Strong support for non-linear development (thousands of parallel branches)
- Fully distributed
- Able to handle large projects like the Linux kernel efficiently (speed and data size)

Why do we git ?

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.



Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

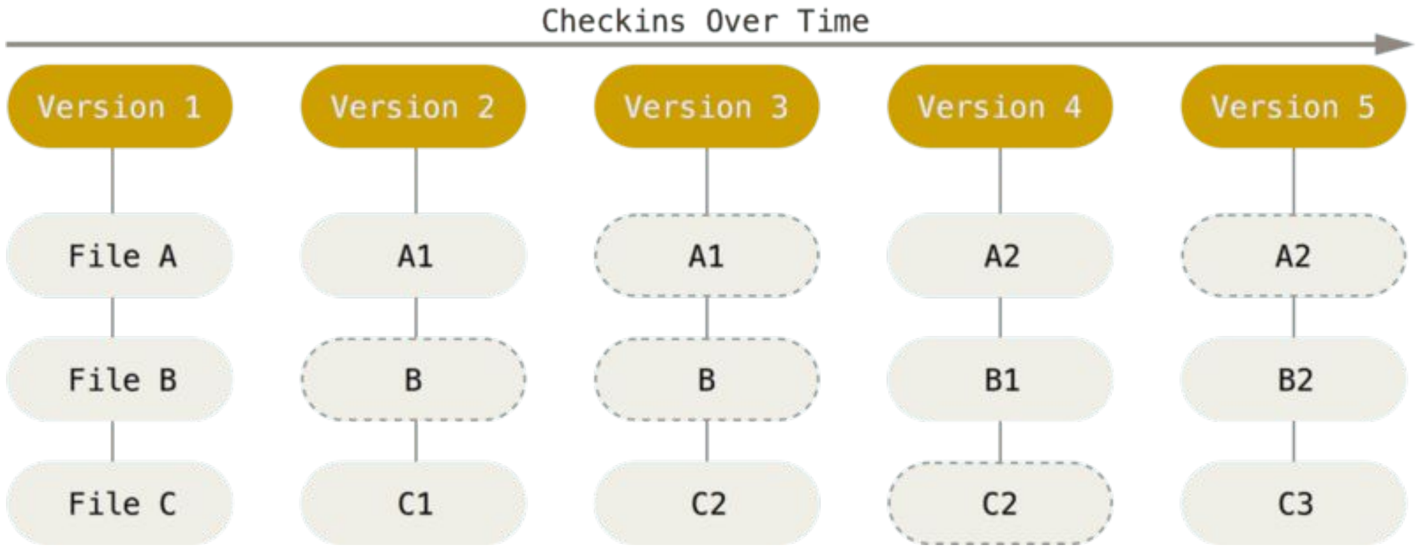
Git Basics

Summary

- *Snapshots, Not Differences*
- *Nearly Every Operation Is Local*
- *Git Has Integrity*
- *Git Generally Only Adds Data*
- *The Three States*

Git Basics

Snapshots, not differences



other systems == diff on files

git == snapshots like a filesystem

Git Basics

Nearly Every Operation Is Local:

For example, to browse the history of the project, Git doesn't need to go out to the server to get the history and display it for you — it simply reads it directly from your local database.

Git Has Integrity

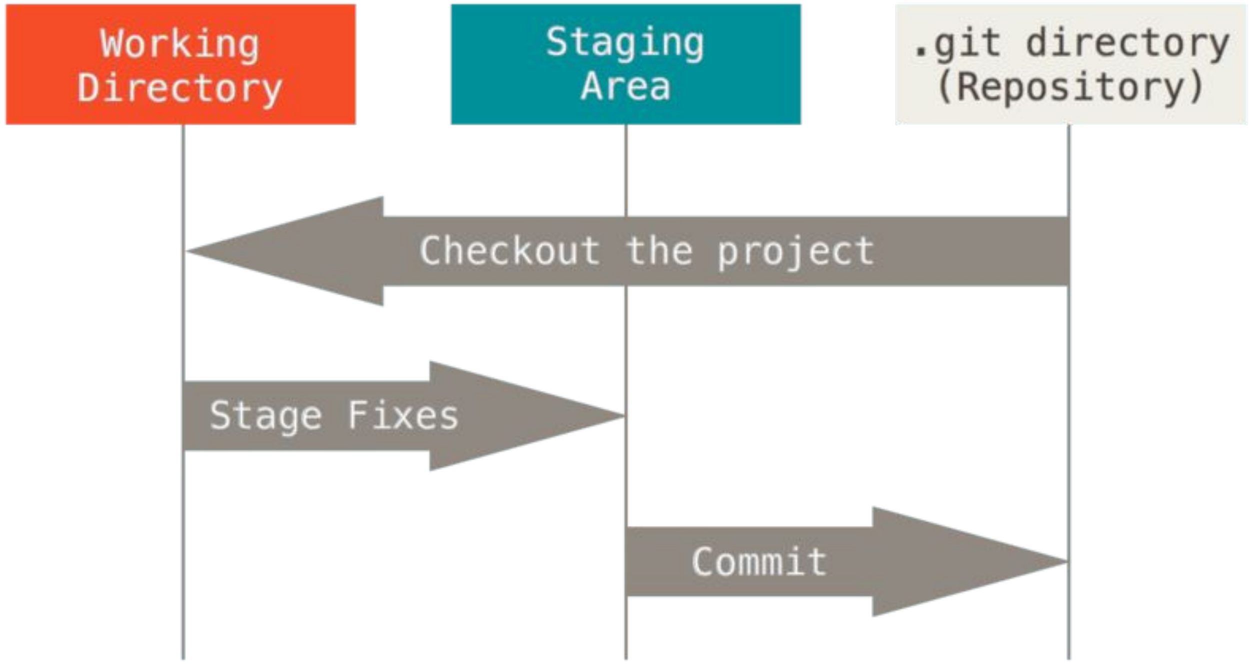
Everything in Git is check-summed before it is stored and is then referred to by that checksum. You will see these hash values all over the place in Git because it uses them so much.

Git Generally Only Adds Data

This makes using Git a joy because we know we can experiment without the danger of severely screwing things up.

Git Basics

The three states



Basic Workflow

The basic Git workflow goes something like this:

1. You modify files in your working tree

2. You selectively stage just those changes you want to be part of your next commit, which adds *only* those changes to the staging area

3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory

Git basics

https://git-scm.com/book/en/v2/Getting-Started-Git-Basics

Git CLI

- Plumbing and Porcelains
- Basic snapshotting
  - `add`
  - `status`
  - `diff`
  - `commit`
  - `reset`, `rm`, `mv`

Git CLI

commit

Running git commit checksums all project directories and stores them as tree objects in the Git repository.

Git then creates a commit object that has the metadata and a pointer to the root project tree object so it can re-create that snapshot when needed.

Git CLI

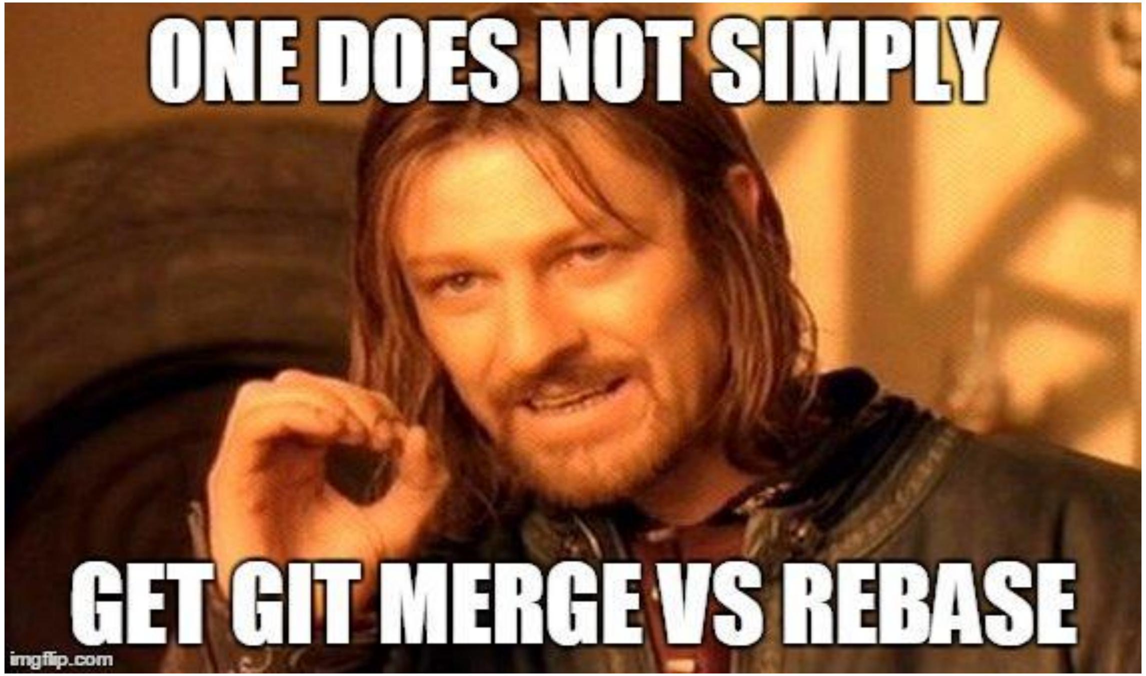
branch == pointer

branch != duplication

Branching

```
          A---B---C topic
         /
    D---E---F---G master
```

MERGE || REBASE



Git CLI

Branching / **Merging**

```
      A---B---C topic
     /         \
D---E---F---G---H master
```

Git CLI

Branching

```
          A---B---C topic
         /
    D---E---F---G master
```

Git CLI

Branching / **Rebasing**

```
                A'--B'--C' topic
               /
  D---E---F---G master
```

Git checkout

Updates files in the working tree to match the version in the index or the specified tree. If no paths are given, git checkout will also update HEAD to set the specified branch as the current branch.

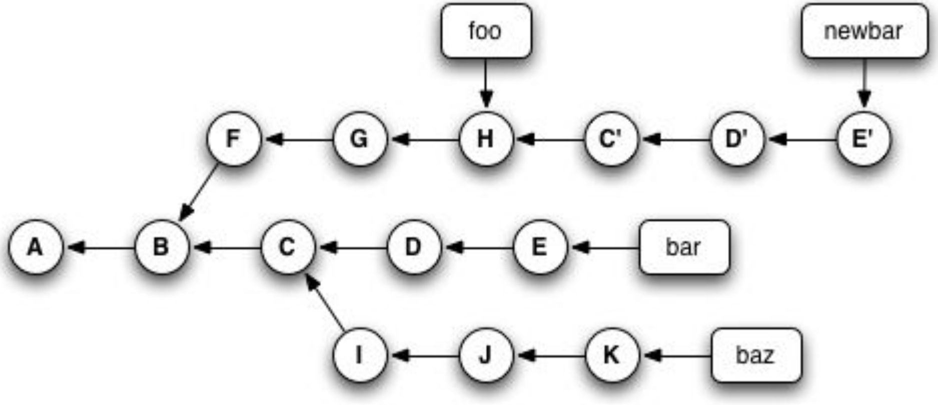
- git checkout <branch>
- git checkout -b
- git checkout .
- git checkout -- <file1>

Stash

Use git stash when you want to record the current state of the working directory and the index, but want to go back to a clean working directory. The command saves your local modifications away and reverts the working directory to match the HEAD commit.

- git stash push
- git stash show
- git stash pop / apply
- garbage collected 90 days

Cherry-pick



Handling remote

- Fetch
- Pull
- Push
- Remote

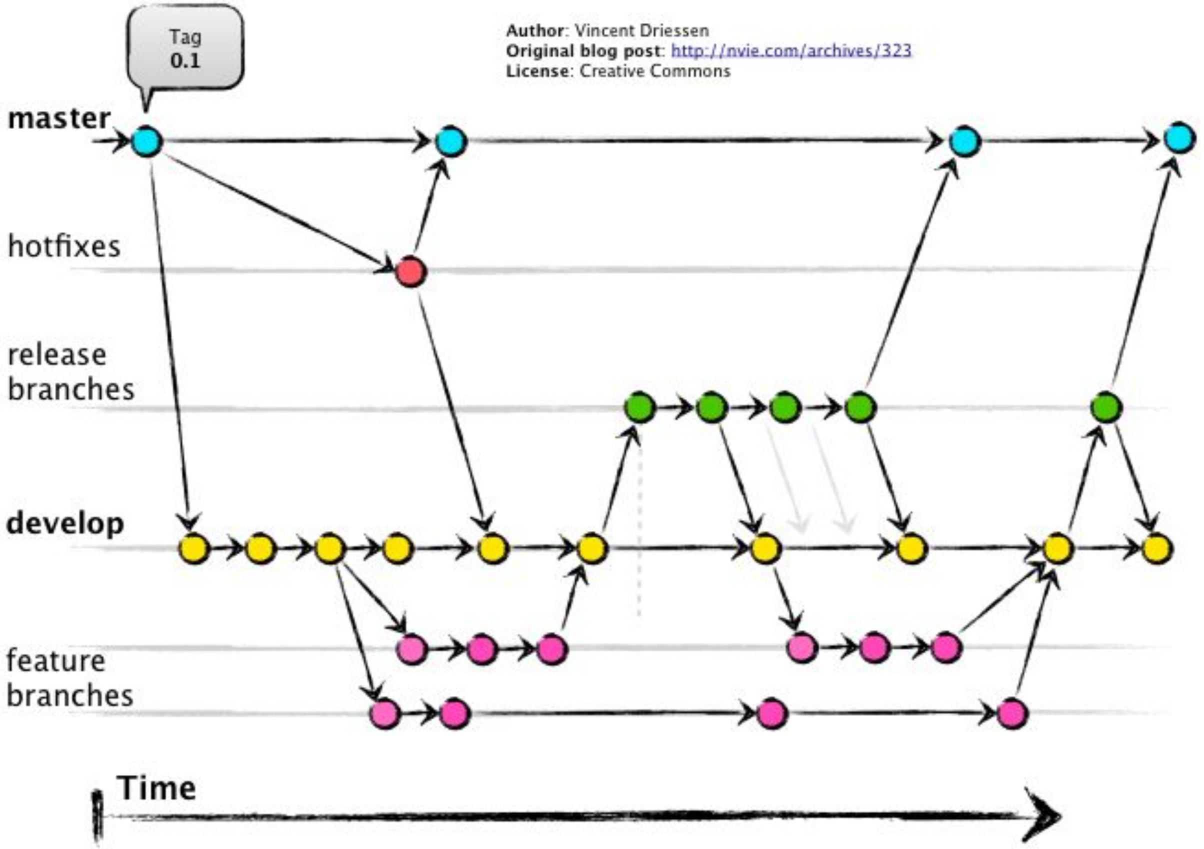
Git cheatsheet

https://ndpsoftware.com/git-cheatsheet.html

Best practices

- What is a good commit message ?
- What is a good workflow ?
- Rebase or merge ?
- Atomic commit ?
- Should I squash my commits ?
- Never push force on master

Git flow


Tag
0.1
Author: Vincent Driessen
Original blog post: http://nvie.com/archives/323
License: Creative Commons
master
hotfixes
release
branches
develop
feature
branches
Time

Sources & links

https://git-scm.com

https://giphy.com