Aha! Moments When Learning Git

Git is a fast, flexible but challenging distributed version control system. Before jumping in:

- Understand regular version control
- Understand distributed version control

Along with a book, tutorial and cheatsheet, here are the insights that helped git click.

There's a staging area!

Git has a staging area. **Git has a staging area!!!**

Yowza, did this ever confuse me. There's both a repo ("object database") and a staging area (called "index"). Checkins have two steps:

- `git add foo.txt`
  - Add foo.txt to the index. It's not checked in yet!
- `git commit -m "message"`
  - Put staged files in the repo; they're now tracked
  - You can "`git add --update`" to stage all tracked, modified files

Why stage? Git's flexible: if a, b and c are changed, you can commit them separately or together.

But now there's two undos:

- `git checkout foo.txt`
  - Undo local changes (like svn revert)
- `git reset HEAD foo.txt`
  - Remove from staging area (local copy still modified).

Add and commit, add and commit -- Git has a rhythm.

Branching is "Save as..."

Branches are like "Save as..." on a directory. Best of all:

- Easily merge changes with the original (changes tracked and never applied twice)
- No wasted space (common files only stored once)

Why branch? Consider the utility of "Save as..." for regular files: you tinker with multiple possibilities while keeping the original safe. Git enables this for directories, with the power to merge. (In practice, svn is like a single shared drive, where you can only revert to one backup).

Imagine virtual directories

I see branches as "virtual directories" in the .git folder. While inside a physical directory (c:\project or ~/project), you traverse virtual directories with a checkout.

- `git checkout master`
  - switch to master branch ("cd master")
- `git branch dev`
  - create new branch from existing ("cp * dev")
  - you still need to "cd" with "git checkout dev"
- `git merge dev`
  - (when in master) pull in changes from dev ("cp dev/* .")
- `git branch`
  - list all branches ("ls")

My inner dialogue is "change to dev directory (checkout)... make changes... save changes (add/commit)... change to master directory... copy in changes from dev (merge)".

The physical directory is a scratchpad. Virtual directories are affected by git commands:

- `rm foo.txt`
  - Remove foo.txt from your sandbox (restored if you checkout the branch again)

- `git rm foo.txt`
    - Remove foo.txt from current virtual directory
    - Gotcha: you need to commit that change!

Know the current branch

Just like seeing your current directory, **put the current branch in your prompt!**

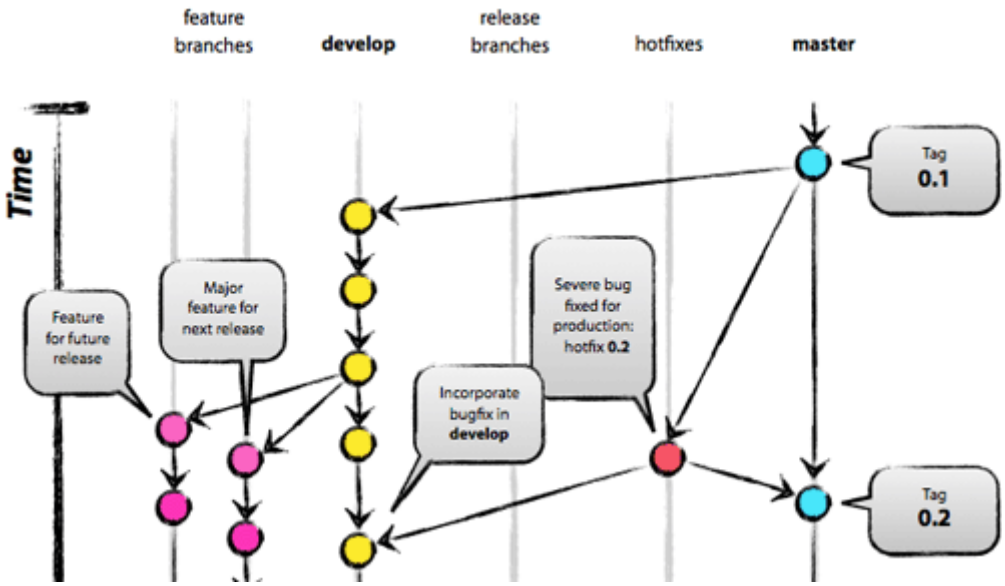
In my .bash_profile:

```
parse_git_branch() {
    git branch 2> /dev/null | sed -e '/^[^*]/d' -e 's/* (.*)/(1)/'
}

export PS1="[33[00m]u@h[33[01;34m] W [33[31m]$(parse_git_branch) [33[00m]$[33[00m] "
```

Visualize your branch structure

Git leaves branch organization to you. Nvie.com has a great branch strategy:



- Have a mainline (master). Mentally it's on the far right.
- Create branches (master -> dev) and subbranches (dev -> featureX). The further from master, the crazier.
- Only merge with neighbors (master -> dev -> feature X, or featureX -> dev -> master)

Stay sane by choosing a branch layout up front. I have a master tracking a svn project, and dev for my own code. In general, master is clean so I can branch anytime for one-off fixes.

Understand local vs. remote

Git has local and remote commands; seeing both confused me ("When do you checkout vs. pull?"). Work locally, syncing remotely as needed.

Local data

- `git init`
  - create local repo
  - use git add/commit/branch to work locally

Remote data

- `git remote add name path-to-repo`
  - track a remote repo (usually "origin") from an existing repo
  - remote branches are "origin/master", "origin/dev" etc.
- `git branch -a`
  - list all branches (remote and local)
- `git clone path-to-repo`
  - create a new local git repo copied from a remote one
  - local master tracks remote master
- `git pull`
  - merge changes from tracked remote branch (if in dev, pull from origin/dev)
- `git push`
  - send changes to tracked remote branch (if in dev, push to origin/dev)

Why local and remote? Subversion has central checkins, so you avoid committing unfinished work. With git, local commits are frequent and you only push when ready.

GUIDs are GOOD

Git addresses information by a hash (GUID) of its contents. If two branches are the same, they have the same GUID (and vice versa).

Why's this cool? We can create branches independently, merge them, and have a common GUID. No central numbering needed. Usually, we just compare the first few digits: "Are you on a93?".

Tips & Tricks

For your .gitconfig:

```
[alias]
        ci = commit
        st = status
        co = checkout
        oneline = log --pretty=oneline
        br = branch
        la = log --pretty="format:%ad %h (%an): %s" --date=short
```

There are some GUI tools for git, but I prefer to learn via the command line. Git is opinionated software (which I like), and analogies help me understand its world view.

Frequently Asked Questions

Your question

Suggest question

Other Posts In This Series