Devops – modern software development practices

Github

* Key principle: Release Continuous Improvement (Remove bugs!) (adds features)

Manage versions:

* Git manages versions of project.
* Each version: Commit. (manage small changes)
* Each commit – snapshot of the entire project.
* Each unique file – saved only once – efficient.
* Review history, undo a change.

Branches:

* Independent line (default: master)
* create separate branch – work without affecting master.

Pull requests:

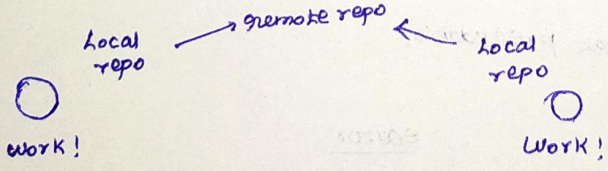
* Branches – merged using pull request (part of master)!
* discuss, review, approve – changes!

Git:

* Complete history tracked & available
* workflows, collaboration, quality – Comm & Review.
* Agility – easily test, fix, undo changes!

Content: → Source Code etc..
Automated tests

Distributed Version Control System



* Git – distributed version control system.
* open source software
* Adopts – many projects & workflows.

Command line

* Most wanted!, automable!, fast & easy!,

Sourcetree: ★ You don't — not so predestined! — UI!
(go both ways!)

Cmd

git [command] [--flags] [argume]

$ git status ?
on branch master
$ git status --short
$ git add file.txt

$ git help init
$ git help
$ git <command> -h → specific

★ -/-- Set flag — change command's behaviour
★ | or
★ [optional]
★ <placeholder> → replace with actual value.
★ ( ) grouping. ? Dis ambiguate the command
... multiple occurences possible

git fakeCommand (-p#--patch) [<id>] [--] [<paths>...]

Configure user info & default editor!

git config [--local | --global | --system] <key> [<value>]

#set user name and email
$ git config --global user.name "pat"
$ git config --global user.email "pat@example.com"

--system → all users on PC
--global → Every repo on PC
No flag → local applies (highest precedence)

$ git config.user.name
Pat
$ get config.user.email
Pat@email.com

editor

$ git config --global core.editor nano.

git locations

Working tree – Single Commit's directories & files.

Staging area/Index – files (planned for next Commit)

Local repo – all commits of the project (version history)

Project directory → working tree, staging area, local repo [In a single directory]

---

* ./git → hidden (local repo, staging area)

*

Remote repo: datacenter/cloud – contains the commits of the project

---

create a local repo

* Create local repo which automatically include working tree & staging area.
* New → Local repo → destination path (project path), name, Git.

---

*

using cmd

* mkdir → make directory
* cd repoa → go to that directory
* git init

```
$ mkdir repos
$ cd repos
repos $ mkdir myproj
myproj $ git init
Initialized empty git repo
myproj $ ls -a
.  ..  .git
```

Commit to a local repo – using Sourcetree

* File Status – empty (initial)
  – Untracked (new file)

① Stage! → point of next Commit

② Commit – Screenshot saved in local repo!
"with commit message"

Change – tracked – Staged file
(State: modified)

History

---

using cmd

$ touch fileA.txt

$ git add _____ (staging)

$ git add . (all) → Don't use very often

$ modified: previously staged - modified

Add directories

```
git add dirA
```

>> git add hello.txt

>> echo "hello I'm" > hello.txt

>> git status

Commit

$ git commit -m "Initial Commit"

$ git status

$ git log

$ git log --oneline (condensed version)

$ git log --oneline -2 (last 2 commits)

Remote repository

* Cloud - Source eg Github
  Bitbucket
  GitHub

* Remote repo : base repo
* end with .git

→ behind : git init --base (created!)

Import - Push to a repo

* Clone / add the repo!
* clone - local copy of remote repo! (synchronize repo!)

Remotes : origin → Substitute for URL
Setting → Remote
origin : hyperlink

→ get URL of repo → clone → URL!

Add a remote repo to a local repo

→ Assumes : existing local repo (Start from left)!
(Synchronise - Job done)!

Starting scenario → Clone
→ Commits in local repo (add to remote)

Remotes → New Remote → name (origin), URL,
Host type, URL, username!

Allows fetch, Pull, Push

* All commits - belong to a branch (default: master)

master Ⓐ—Ⓑ—Ⓒ

Local (Track)

Push to remote (push → master →

Track: relation management b/w remote & local in future!

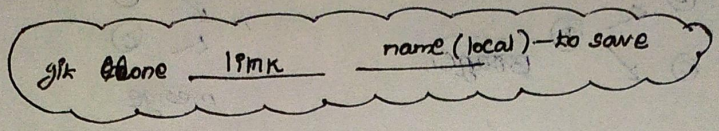
master | origin/master

Command line

Clone → local copy of a remote repo.

git clone <url / to / projectname.git> [local projectname]

* git remote --verbose → info about remote repo associated with local

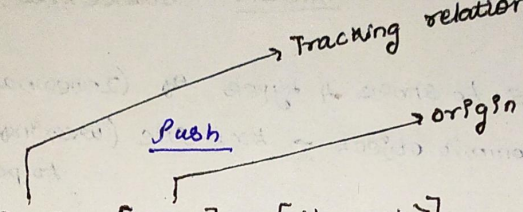


Add remote repo

1) git remote add <name> <url>

git remote add origin <link>

git remote -v



git push [-u] [repo] [<branch>]

-u - track

→ git push -u origin master

Adding commits - for a cloned repo!

repo $ git clone _____ name

$ cd repoa

$ echo "# My project" >> README.md.

$ git add README.md

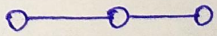
$ git commit -m "Initial Commit" → local repo

$ git push -u origin master → remote repo.

Graph model

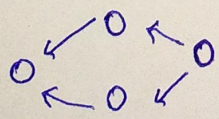
* Directed Acyclic graph!

nodes, edges



3 nodes, 2 edges!

* directed, non-directed
* Acyclic - No cycles - can't path - starts & end at same path!



DAG

Git's DAG

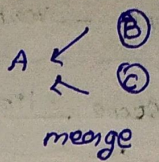
* Arrows point at a commit's parent

commits & relationship b/w them

↳ "history"

Branch 1 (B) ← (D) Branch

Branch 2 (C) ↙



No arrows (↓ vertical order)

$ git log --oneline --graph

Git IDS - Source tree!

Git objects

* Git uses objects to store 4 types of (Internally) things.

1. Commit object - txt file (user info, commit message, references) to parents/parent/root tree.

Commit & tags

2. Annotage tag - reference to a specific commit
3. Tree - Directories & filenames in the object!
4. Blob - Content of a file - managed by GIT.

GID IDS

→ name of a Git object - 40 char hexadecimal string - also known as object ID, SHA-1, hash & checksum

→ Secure hash algorithm - 1 (SHA 1) - unique.

Avalanche: even small change - creates greater change in SHA-1

Shortening GIT IDS

→ Shows first portions - 7 (or) 10 characters

Cmd

* Git object: $ git log

Create Sha-1

* git hash-object <file> → create an SHA-1 for any content
* Git-hash object → example of lowlevel - plumming command!

(Scripting - useful)

Shorten

$ git log --oneline.

$ git log → full ID

$ git show 483d (atleast 4 char)

References

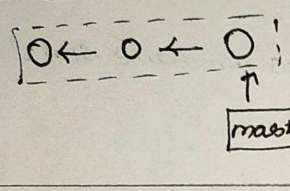
Branch label & head reference:

* Reference – user friendly – points to Sha 1 / other reference (symbolic reference)

eg: HEAD → master

Master → default name of the main branch (independant path).

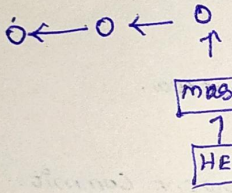
("points to the tip of the branch")



* All the commits belong to master branch even though the master branch label is at the end of the branch.

↳ points to the most recent commit in the branch!

* Tiny branch label – branches – extremely simple & use few resources.
* HEAD → reference to the current commit. usually points to the branch label of the current branch. (only one current commit!) → one head per repo!



Tags

Tags: Reference to specific commit – user friendly label – Different from branch label – which move along with each commit.

Use: mark important commits (version 1.0)

Tag → V0.1 (or) working copy parent (Recent)
(Specify Reference).

* Adv options: * move existing tags = can't create tag with existing name.
* Annotated tag – used by Source tree (light weight: not recommended – less data)
* Sign message: add tag message (for tags like commit message)
* Push → push tag!

tags → push to → origin

Source tree!

Add tags → push to → commits.

Cmd - References

* Branch labels, Head.
* Reference: User friendly name → points to SHA1 / other reference (Symbolic ref)

HEAD → master

Uses references instead of SHA1

$ git log
$ git show HEAD / git show 5dig SHA1 / git show full SHA1 / git show master

'HEAD, master ref — same all points to same SHA1'

MASTER default name of main branch - Independent path

Branch label: points to most recent commit — Here: master: branch!
HEAD points to branch label → At the tip of the branch!

Local branch references — IN .git/refs/heads

$ cd .git
$ ls
$ cd refs
$ cd tags
$ ls
master → only one branch label

'only one branch as of now'

$ cat master
SHA1 → Contents of master.

* HEAD - Reference to current Commit.
* points to branch label — one head per repository!

.git/HEAD

$ cat HEAD
ref: refs/heads/master

∴ Content: reference to master

only one head per repo in .git

Referencing prior Commits with ~ and ^

~ Tilde — to git tds & references:

~ (or) ~1 = parent
~2 (or) ~~ = parent's parent.

$ git log --oneline --graph

$ git show HEAD

Shows commit info for current object (SHA1 value beg with 1ef16ac)

$ git show HEAD~

current Commit's parent whose SHA1 begins with 1ef16ac.

$ git show master~3 [3 commits away from current one].

$ git show e0cb6c5~~~ [3 "] whose sha1 begins with e0cb6c5

^1 - first parent of the commit

^2 - second parent of a merge commit

^^ - first parent's first parent

→ HEAD ~ ^2 ⟶ parent's second parent

Tag - reference to specific Commit

Types:

* Light weight - A simple reference to a commit (Branch label/HEAD)
* Annotated - Full object (Similar to Commit object) - too much info!

$ git tag [View all tag]

v0.1

v1.0

$ git show v0.1 (Shows Commit associated with that tag).

$ git tag <tagname> [<Commit>]

with tagname — Commit to be tag (option - else HEAD points will be tagged)

$ git tag v0.1 HEAD^

parent of the current Commit

tag ⟶ Annotated tag!

↙ lightweight

git — branches

Benefits:

* Fast, easy
* Enable experimentation
* Enable team development
* Support multiple versions project

* Short lived branches : topic/feature branches - one small change to the project
* Live long (long lived) : master, develop, release etc..

Create

* Just new branch label Branch → name → create

checkout

*updates the HEAD references (master branch (now) → checkout feature branch → shifts head to feature branch.

* checkout - updates working tree!

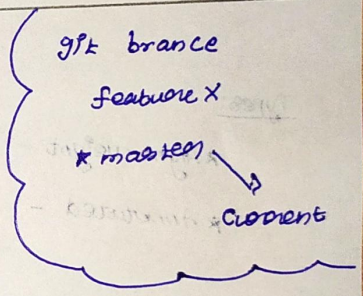
git branch <name>

* go to previous version: detatch head (previous)!

merging

$ git branch featurex
$ git checkout featurex → Changehead
$ git checkout -b featurex
$ git branch -d featurex (only when a topic branch - merged)!

Combines both comm...



Unmerged work

* error! (force delete → Dangling → garbage collect)

Undo - accedental Branch delete

git reflog returns a local list of recent HEAD commits

$ git branch -D featurex
$ git reflog
$ git checkout -b featurex 434dfad

Merge