# Git

Refresher Supplementary

# LIST OF VCS

| | | |
|---|---|---|
| Local only | Free/open-source | RCS (1982) · SCCS (1972) |
| | Proprietary | PVCS (1985) · *QVCS* (1991) |
| Client–server | Free/open-source | CVS (1986, 1990 in C) · CVSNT (1998) · QVCS Enterprise (1998) · Subversion (2000) |
| | Proprietary | AccuRev SCM (2002) · ClearCase (1992) · *CMVC* (1994) · Dimensions CM (1980s) · *DSEE* (1984) · Endevor (1980s) · Integrity (2001) · Panvalet (1970s) · Perforce Helix (1995) · SCLM (1980s?) · Software Change Manager (1970s) · StarTeam (1995) · Surround SCM (2002) · Synergy (1990) · Team Concert (2008) · Team Foundation Server (2005) · Visual Studio Team Services (2014) · Vault (2003) · *Visual SourceSafe* (1994) |
| Distributed | Free/open-source | *ArX* (2003) · BitKeeper (1998) · *Codeville* (2005) · Darcs (2002) · *DCVS* (2002) · Fossil (2007) · Git (2005) · *GNU arch* (2001) · GNU Bazaar (2005) · Mercurial (2005) · Monotone (2003) · *SVK* (2003) · Veracity (2010) |
| | Proprietary | *TeamWare* (1990s?) · Code Co-op (1997) · Plastic SCM (2006) · Team Foundation Server (2013) · Visual Studio Team Services (2014) |
| Concepts | | Branch · Fork · Changeset · Commit (Gated commit) · Interleaved deltas · Delta compression · Data comparison · Merge · Repository · Tag · Trunk |

Source - https://en.wikipedia.org/wiki/Distributed_version_control

# LIFE WITHOUT VCS

- A bug fixed in a previous version reappears in the latest version!
- Customer is sent a release with wrong version of files!
- Code developed does not match with documented design!
- Changes made to the latest version vanishes!
- Someone made some changes in the code, no one knows where!
- Continuous changes are being requested by the client, how to control?
- All the code files got deleted by mistake and there are no backups!
- Nobody knows the status of the work products under development

# DVCS VS CVS

## Bare vs non bare repo of git

### Centralized

A central repository contains all the code
Examples: CVS, Subversion,...

### Distributed

Each user has its own repository
Examples: mercurial, git, ...

A "bare" repository in Git just

- contains the version control information and no working files (no tree) and
- it doesn't contain the special .git sub-directory.
- Instead, it contains all the contents of the .git sub-directory directly in the main directory itself.

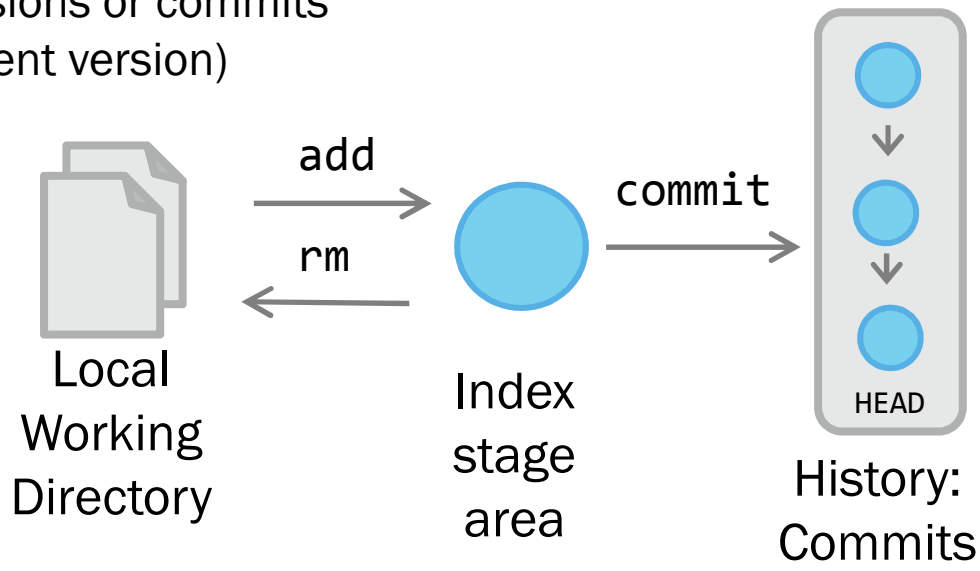
A "non-bare" repository in Git has

- a bunch of working files (the tree), and
- a hidden directory containing the version control information.

In Git (from version 1.7.0 and above) the repository has to be "bare" (no working files) in order to accept a push.
Convert from non-bare to bare : git clone --bare -l non_bare_repo new_bare_repo

# LOCAL COMPONENTS OF GIT

Local working directory
Index: stage area, sometimes also called cache
History: Stores versions or commits
    HEAD (most recent version)



# BRANCHES

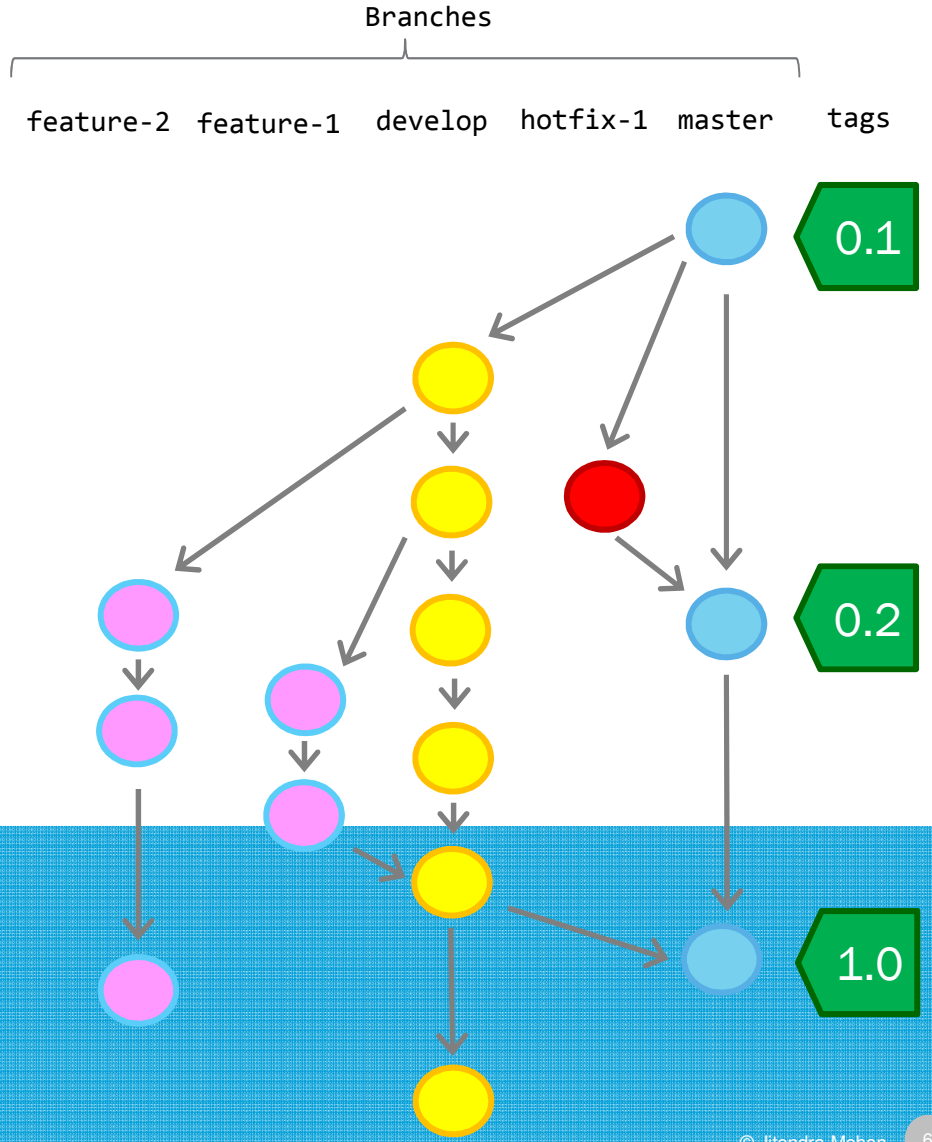
Git facilitates branch management

- master = initial branch

Operations:

- Create a branch (*branch*)
- Switch branch (*checkout*)
- Combine branches (*merge & rebase* )
- Tag branches (*tag*)

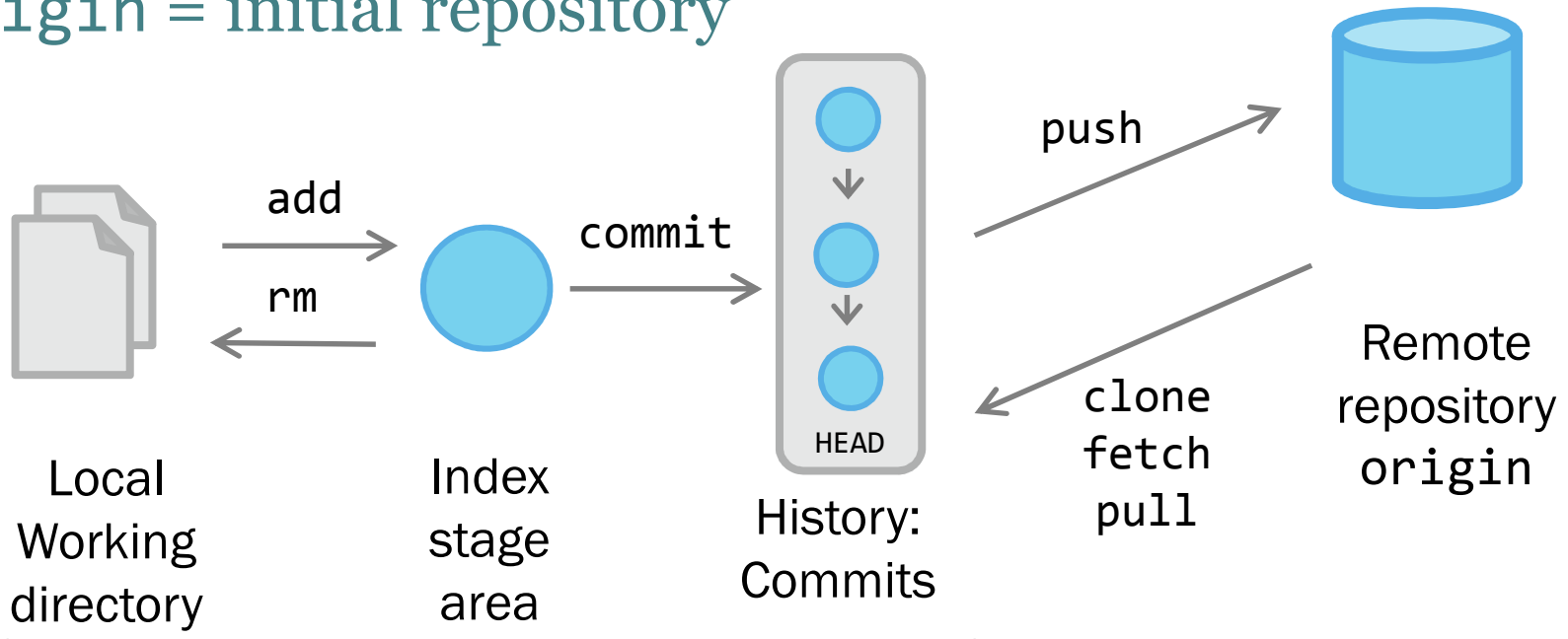
Several branching styles



# WORKING WITH REMOTE REPO

It is possible to connect with remote repositories

`origin` = initial repository

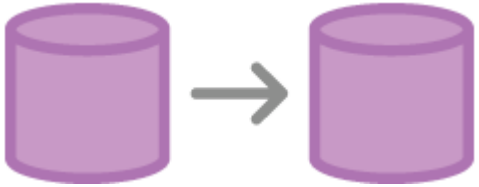


# INIT - CREATE REPOSITORIES

`git init`

- Transforms current folder in a *git* repository
- Creates folder .git
- Variants:
  - `git init <folder>`
    - Creates an empty repository in a given folder
  - `git init --bare <folder>`
    - Initializes a Git repository but omits working directory

# CLONE - CLONE REPOSITORIES



```
git clone <repo>
```

Clones repository <repo> in the local machine

<repo> can be in a remote machine

Example:

```
git clone https://github.com/jiten-igt/addressbook
```

NOTA: Like `init`, this instruction is usually done once

# CONFIG - CONFIGURE GIT

```
git config --global user.name <name>
```

- Declares user name
- Other configuration options:
  - user.email, merge.tool, core.editor, ...
- Configuration files:
  - <repo>/.git/config -- Repository specific
  - ~/.git/config -- Global

# ADD - ADD TO THE INDEX

```
git add <file>
git add <dir>
```

Adds a file or directory to the index

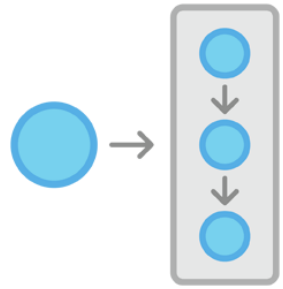
Variants

```
git add --all
```

Index or stage area stores file copies before they are included in the history

# COMMIT

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



- Add files from index to history
  - Creates a new project snapshot
  - Each snapshot has an SHA1 identifier
    - It is possible to assign tags to facilitate management

NOTE: it is convenient to exclude some files from control version Examples: binaries (*.class), `.gitignore` file contains the files or directories that will be excluded

# STATUS - INFO ABOUT INDEX

```
git status
```

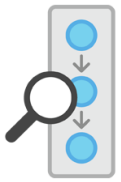
Shows *staged*, *unstaged* and *untracked* files

staged = in index but not in history

unstaged = modified but not added to index

untracked = in local working directory

# LOG - INFO ABOUT HISTORY



`git log`

Shows changes history

Variants

| | |
|---|---|
| `git log --oneline` | 1 line overview |
| `git log --stat` | Statistics |
| `git log -p` | Complete path with diff |
| `git log --autor="expr"` | Commits of one author |
| `git log --grep="expr"` | Searches commits |

`git log --graph --decorate --online`

Shows changes graph

# DIFF - SHOWS DIFFERENCES

`git diff`

Working directory vs index

`git diff --cached`

Index vs Commit

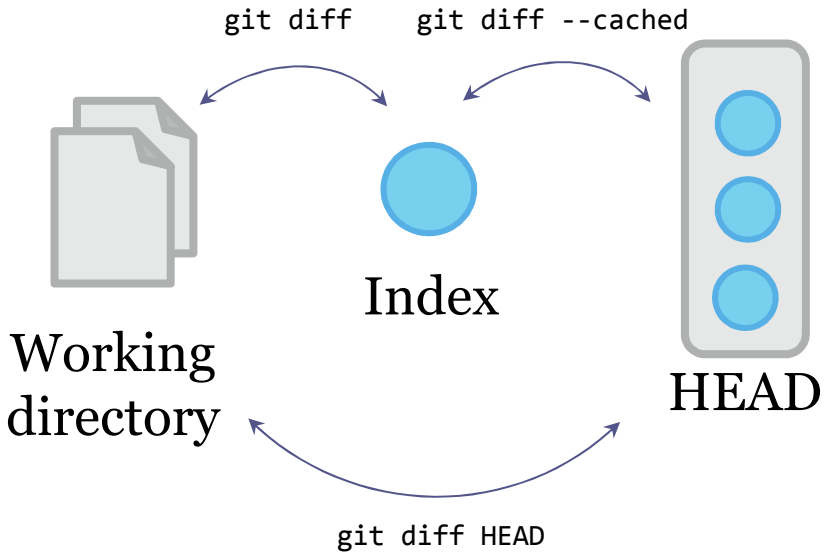
`git diff HEAD`

Working directory vs commit

Some options:

--color-words

--stat

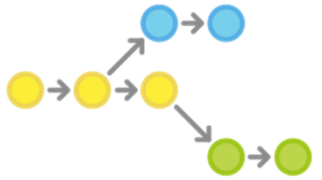


# Branching

```
branch
checkout
merge
```

# BRANCH

## Branch management



`git branch` Shows existing branches

`git branch <r>` Creates branch <r>

`git branch -d <r>` Delete branch <r>

Safe (it doesn't delete if there are pending merges)

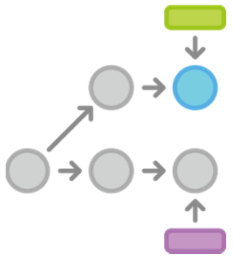
`git branch -D <r>` Delete branch <r>

Unsafe (deletes a branch and its commits)

`git branch -m <r>` Rename current branch to <r>

# CHECKOUT

Change local directory to a branch



```
git checkout <r>
```

Changes to existing branch <r>

```
git checkout master
```

Changes to branch master

```
git checkout -b <r>
```

Creates branch <r> and changes to it

Equivalent to

```
git branch <r>
git checkout <r>
```

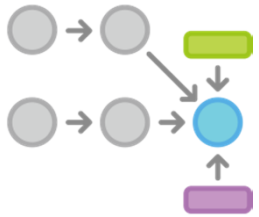
# MERGE

Combine two branches

| | |
|---|---|
| `git merge <r>` | Merge current branch with `<r>` |
| `git merge --no-ff <r>` | Merge generating a commit (safer) |

2 merge types:

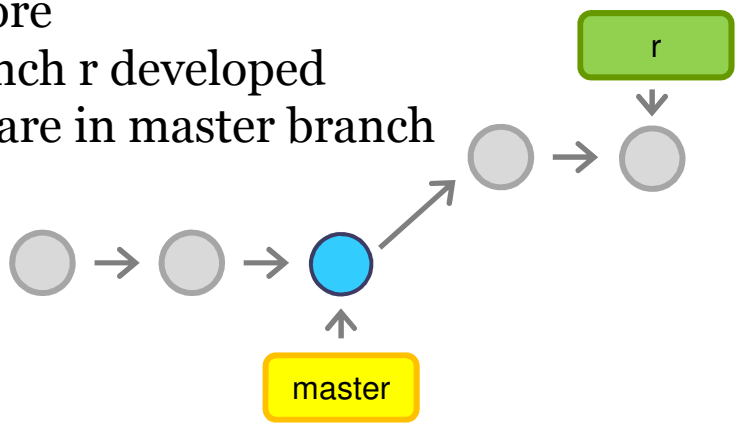
- Merge fast-forward
- 3-way merge



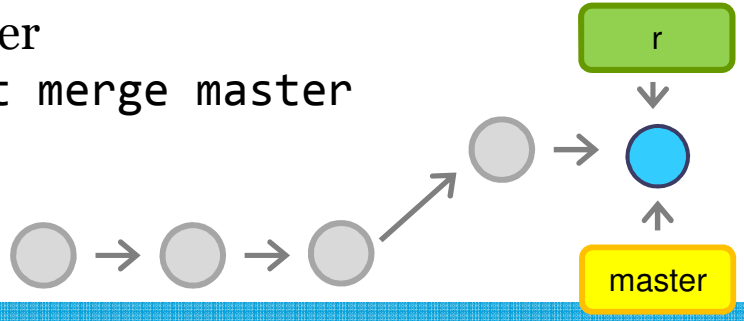
# FAST FORWARD MERGE

When there is lineal path between current branch and the branch to merge

Before
Branch r developed
We are in master branch




After
`git merge master`



# 3 WAY MERGE

When branches diverge and it is not possible *fast-forward*

If there are conflicts:

- Show: `git status`
- Edit/repair:
- `git add .`
- `git commit`