# Introduction to Software Engineering

# Git Tutorial

Rami Sayar @ramisayar

GitHub Talks If you find a mistake, don't hesitate to fork the repository, clone it to your local machine, edit the mistake, git commit -a -m and then press the pull request button to notify me.

## What is git?

Git is a Distributed Version Control System. It is an evolution from local and then central version control systems. Clients don't get the latest version of a code base, instead they checkout a full mirror copy of the repository. With DVCSs like git, you can do local VCS work and still be able to push out the changes in your code base to a central VCS or directly to your peers. Git is a powerful source code management system in use throughout industry. Historically, it was created and designed to handle the incredible quickly paced, highly branched development of the linux kernel.

## Where do I get git?

You can download git for your operating system using your system's package manager or by downloading an installer from http://git-scm.com

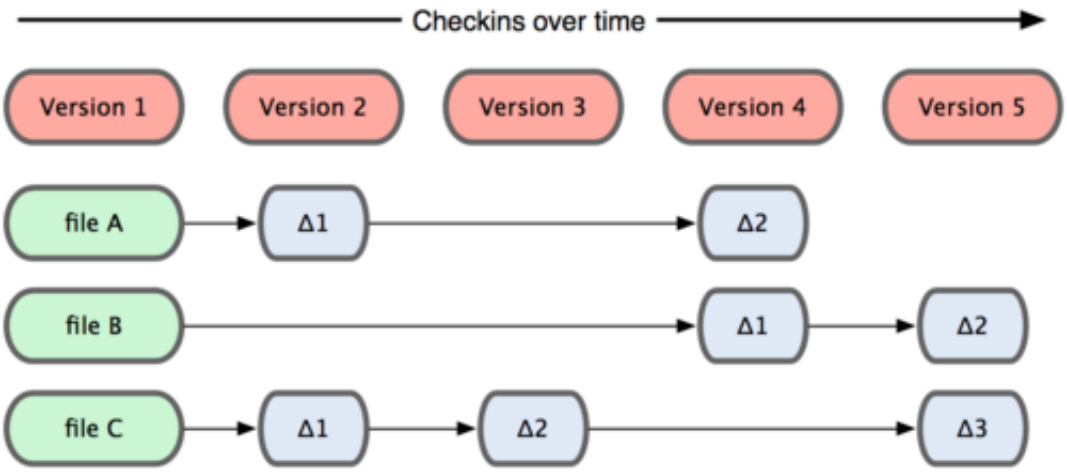
## How do I use git?

The general preferred method of using git is through a terminal. Git comes with an wealth of command line tools. However, it is possible to use git through a

graphical interface but you will be limited to what the tool provides you.

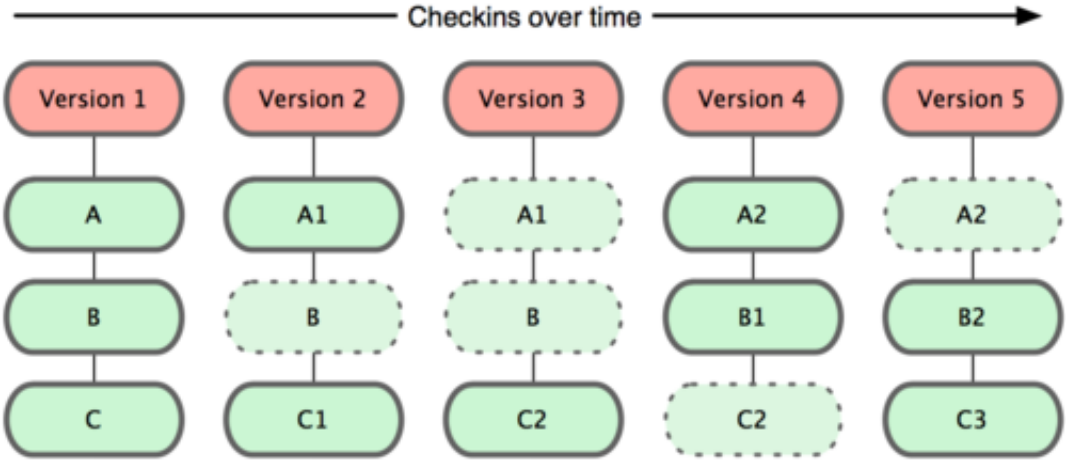
## How does Git Work?

Historically, source control management software would work by tracking the changes to a base file over time.




REFERENCE

Your local repository consists of snapshots of your files. Git treat your repository as if it were a mini filesystem and will store references to the current state.

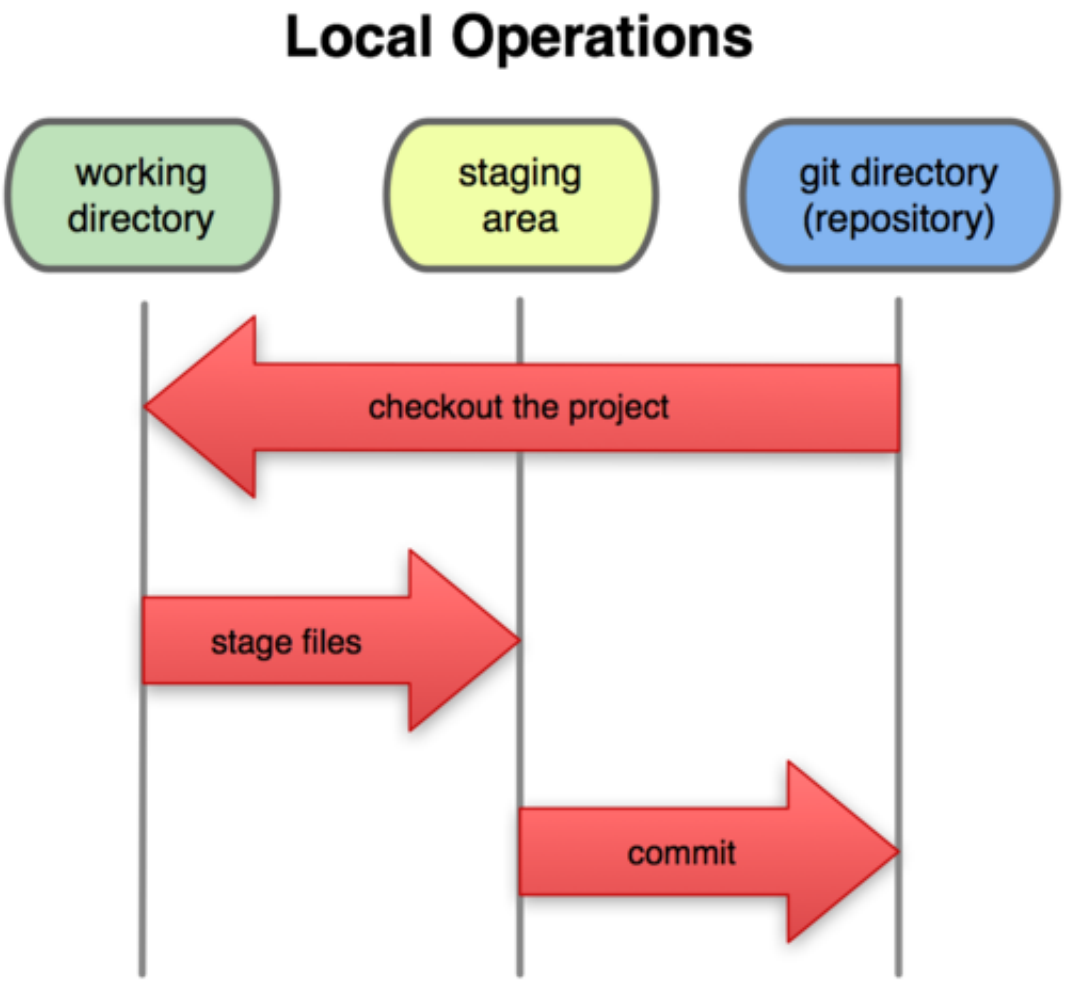



REFERENCE

Everything is local and, as a result, very fast. Your repository can exist entirely within your computer, or be accessible within a LAN, you do not need to get on the Internet. You can literally have git work over network shares (DO NOT DO THIS.)

Everything in git is check-summed and referred to by checksums. You will not be able to hack git to ignore changes in your files. Git will detect every change and it will use hashes everywhere. Git actually refers to files using a checksum as opposed to a file name.

Git almost always simply adds data, so it is very hard to mess up your version control system or be unable to recover data or go back in time.




REFERENCE

# Creating a new repository

```
mkdir sample-project
cd sample-project
git init
```

git init comes with some useful extras such as --template which will allow you to use a template repository and add initial files.

git init --shared allows you to specify repository information if you are sharing a repository with several users.

run `git init -h` for more information

# Adding, Removing and Committing

```
git add path/to/file
git add *

git commit -m "Commit my file."
```

I just committed to the local repository. HEAD.

```
HEAD
HEAD^
HEAD~2
```

Removing files is simple `git rm path/to/file` DONE!

# Branching

```
git branch [branch_name]
git branch -l
git branch -d [branch_name]
git checkout [branch_name]
git checkout -b [branch_name]
```

## Branch Management

```
git branch -v
git branch --merged
git branch --no-merged
```

## Merging

```
git merge [branch_name]
```

Git tries to auto-merge changes. Unfortunately, conflicts arise, so you will have to manually edit the files shown by git. After fixing the file, you add them to the staging area.

```
git add [filename]
```

You can also preview changes using the git diff command.

```
git diff [source_branch] [target_branch]
git diff [filename]
```

## Tagging

To keep track of your software releases, you can use a the tagging feature.

```
git tag
```

Tags exist in two forms, lightweight and annotated. Lightweith tags can not be changd, but annotated tags are full objects that can have messages, be updated, etc…

Lightweight tag:

```
git tag first-version [COMMIT_ID]
git tag 1.0.0 [COMMIT_ID]
```

You can get the commit id by using the log.

Annotated tag:

```
git tag -a 1.0.0 -m "Here is my first version"
```

# Using the Log

```
git log
git log --stat
git log --author=sayar
git log --stat subdirectory
git log --grep='.*foo.*'
git log --since="1 week ago" --until="2 weeks ago"
git shortlog -sn
```

REFERENCE

# Replacing local changes

`git checkout -- [filename]` Replaces changes in the working tree to whatever is in HEAD.

HOWEVER, changes in the staging area and new files are kept. To drop all changes you have to reset your repository.

```
git reset
git reset --soft HEAD^
git reset --hard
```

Undo a commit and redo

```
git commit ...
git reset --soft HEAD^
edit
git commit -a -c ORIG_HEAD
```

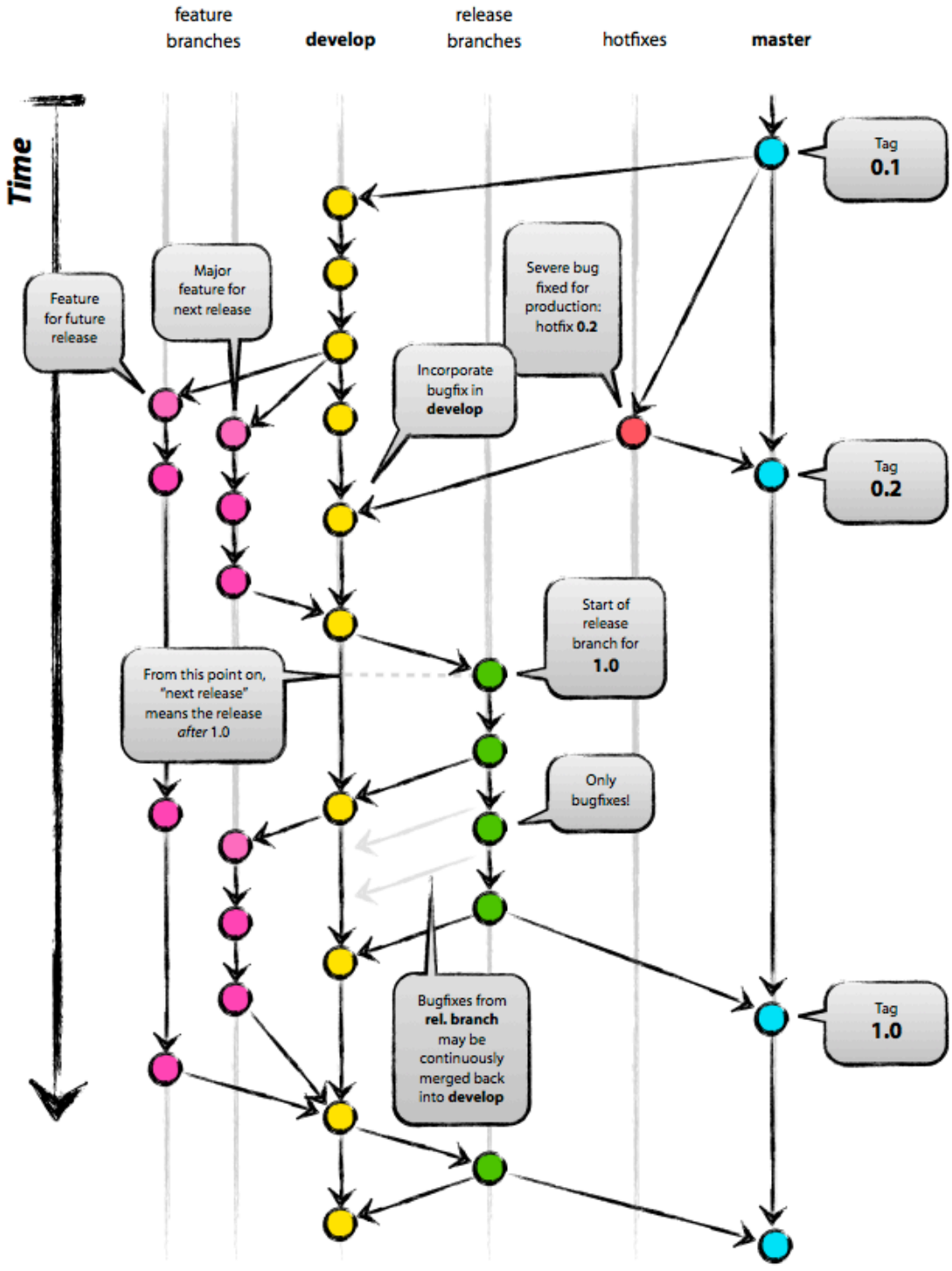
Undo a commit, making it a topic branch

```
git branch topic/wip
git reset --hard HEAD~3
git checkout topic/wip
```

This will still not delete new files you added, so you can recommit them if necessary.

# Branching Workflows

Long-Running Branches - As branches are simply pointers to commits, you can keep running many branches without a great difficulty in merging back.




REFERENCE

Topic Branches - Branches can be short-lived. You can create and use a branch for implementing a single feature or topic. As it is fairly straightforward to merge

back, I highly recommend you do so as it allows you segment your work into manageable chunks.

## Getting an existing repository

```
git clone /path/to/repo
git clone username@host:/path/to/repo
```

Github Example: `git clone git@github.com:username/repo.git`

## Pushing changes to remote repository

```
git remote add origin /path/to/repo
git remote add origin username@host:/path/to/repo

git push [remotename] [localbranch]:[remotebranch]
git push origin master
```

## Fetching

To synchronize repositories with a remote repository you can use fetch.

```
git fetch origin
```

## Merging Remote Branches

```
git merge origin/master
git checkout -t origin/[branch]
```

## Deleting Remote Branches

`git push [remotename] :[remotebranch]`

# Using Pull

If you already have an upstream setting, you can use git pull to do a fetch and merge.

```
git pull [remote] [branch]
git pull origin master
```

# GitHub Pull Request

Pull requests let you notify a repository owner that you've offered a set of changes for a merge. If a pull request is sent, the owner can review, discuss changes, add commits and then merge your set of changes into a branch.

### Fork & Pull

Allows you to clone a repository and then notify the owner of changes you've made in your own fork.

### Shared Repository Model

If you are sharing a repository, you can still use pull requests to discuss and notify each other about changes taking place in your branch before you merge.

GitHub Using Pull Requests

# Pushing and Pulling Tags

Tags are just like other objects, they need to be pushed and pulled.
`git push [remotename] [tagname]`

git pull brings all the tags down.

# Git Stashing

```
git stash
git status
git stash list
```

To apply the last stash: `git stash apply`

To apply an older stash: `git stash apply stash@{2}`

Create a branch from an stash to continue working:
`git stash branch testchanges`

# Git Config

```
git config --global -e

git config --global user.name "Rami Sayar"
git config --global user.email "rami.sayar@gmail.com"

git config color.ui true
git config format.pretty oneline

git config --global alias.co checkout
git config --global alias.br branch
git config --global alias.lg log
git config --global alias.st status
```

# Git Ignore

.gitignore is a file in your repository which informs git to ignore certain files and directories.

GitHub GitIgnore Repostiroy

# Useful Hints

built-in git GUI `gitk`

Interactive adding `git add -i`

# Advanced Topics

- Git Branch Rebasing
- Git Tools - Revision Selection
- Git Tools - Rewriting History
- Git Tools - Submodules
- Git Tools - Subtree Merging

# References


- git - the simple guide by Roger Dudler
- Pro Git by Scott Chacon
- github:help