Introduction to Git

Rishita Narayan
Developer
rishita.narayan@thoughtworks.com

/thoughtworks

Collect Participant github account and add as collaborator

/thoughtworks

Agenda

What is Git

Understand problem statement, why we need Git/version control, What makes Git so popular, Understand local and Remote Repo

- Why Git
- What is Git
- Local vs Remote Repo
- GUI Tools

Git Basics

Learn three states in GIT, Understand Basic commands for setup, making changes, Understanding Commit

- How Git Works
- Basic commands
- Making Changes to a file
- Understanding Commits

Undoing things

View Commit history, Learn different cases when to use Amend, Revert and reset commands.

- Viewing Commit History
- Amend commit
- Revert
- Reset

Branching, Merging

Understanding Branching, use Cases for branching, understand Merge vs Rebase, explore workflows

- Branching
- Merging
- Rebase
- Workflows

Goals for this session

In today’s practical session, we will be learning git.

Understand what is Git and why we use it

Understand key concepts
- Local vs Remote Repositories
- Working directory,Staging area, Local Repo, Stashing

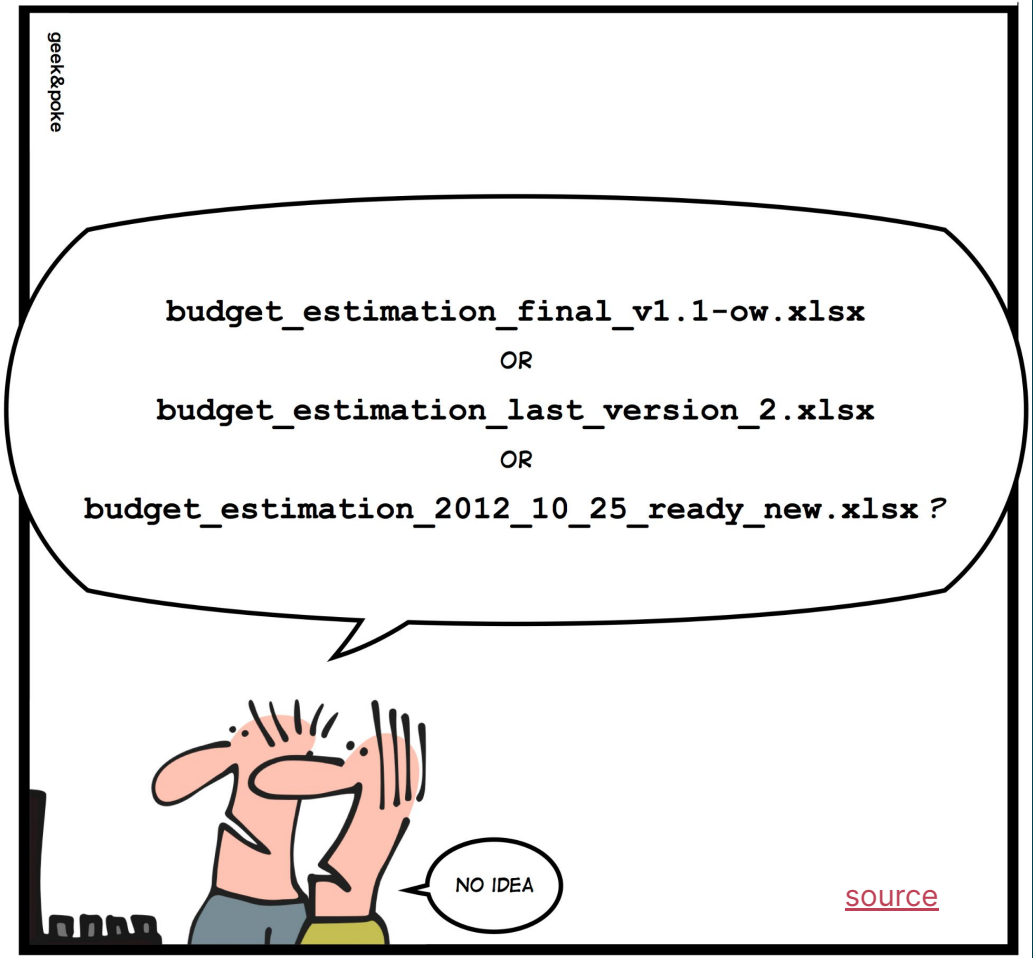
Able to use basic commands
- clone, pull,
- add, commit, push
- log

Know the resources, where to look for

Why Do we need Git

/thoughtworks

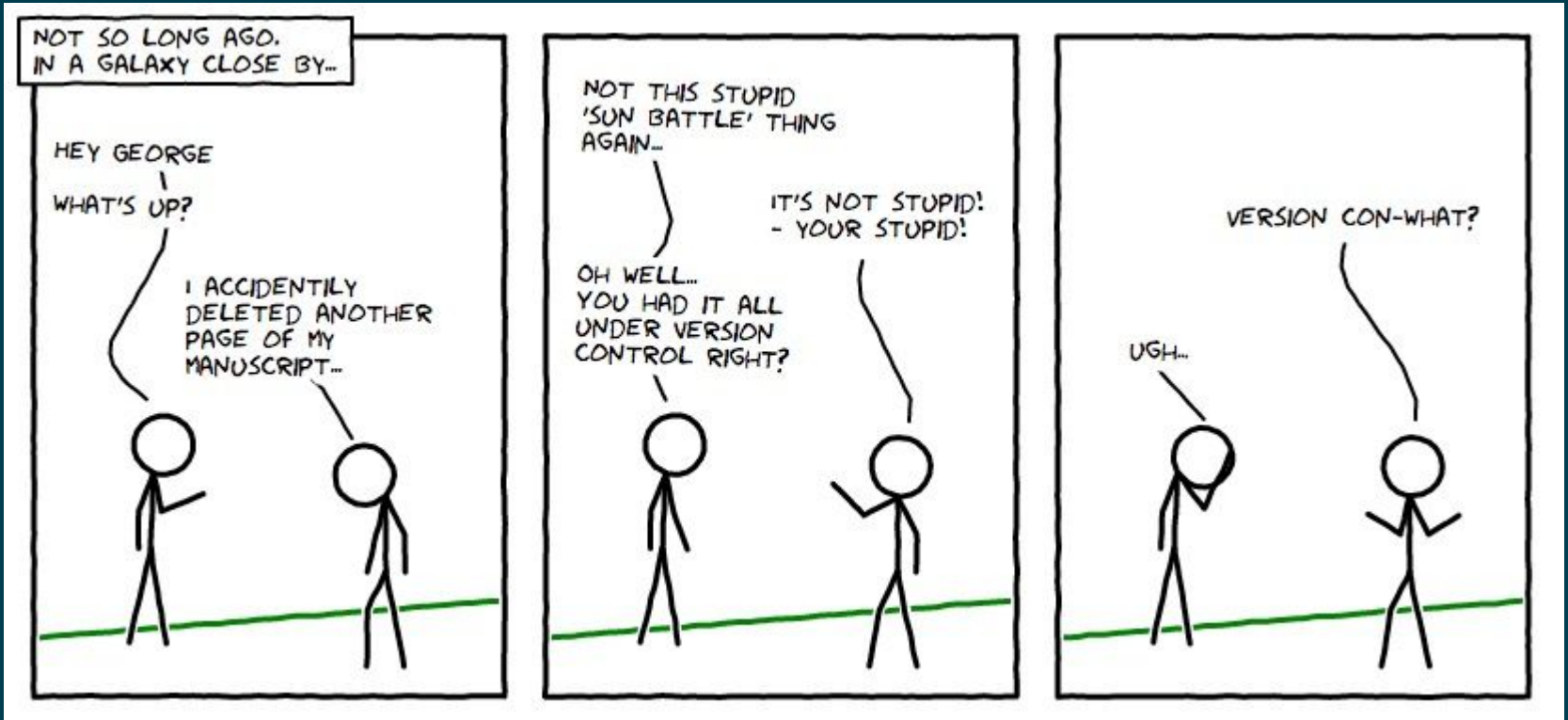
SIMPLY EXPLAINED



source

VERSION CONTROL

/thoughtworks



source

/thoughtworks

What is Git

/thoughtworks

Git is a decentralised Version control System

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

It allows you to revert selected files back to a previous state

revert the entire project back to a previous state

compare changes over time

Check who modified what/which file, when, whose change introduced a bug, who can help to fix

Why Git is so popular

Git is most popular Version control system, mainly due to its highly scalable nature. Some of the key points like Simple design, Strong support for non-linear development (thousands of parallel branches), Fully distributed, Able to handle large projects make it very efficient

Fast

Most operations are local

Reliable

No changes are ever lost

Supports NonLinear Dev

Many workflow options

Easy to Use

Wider Adoption

Active Community

Lot of resources available

Alternatives

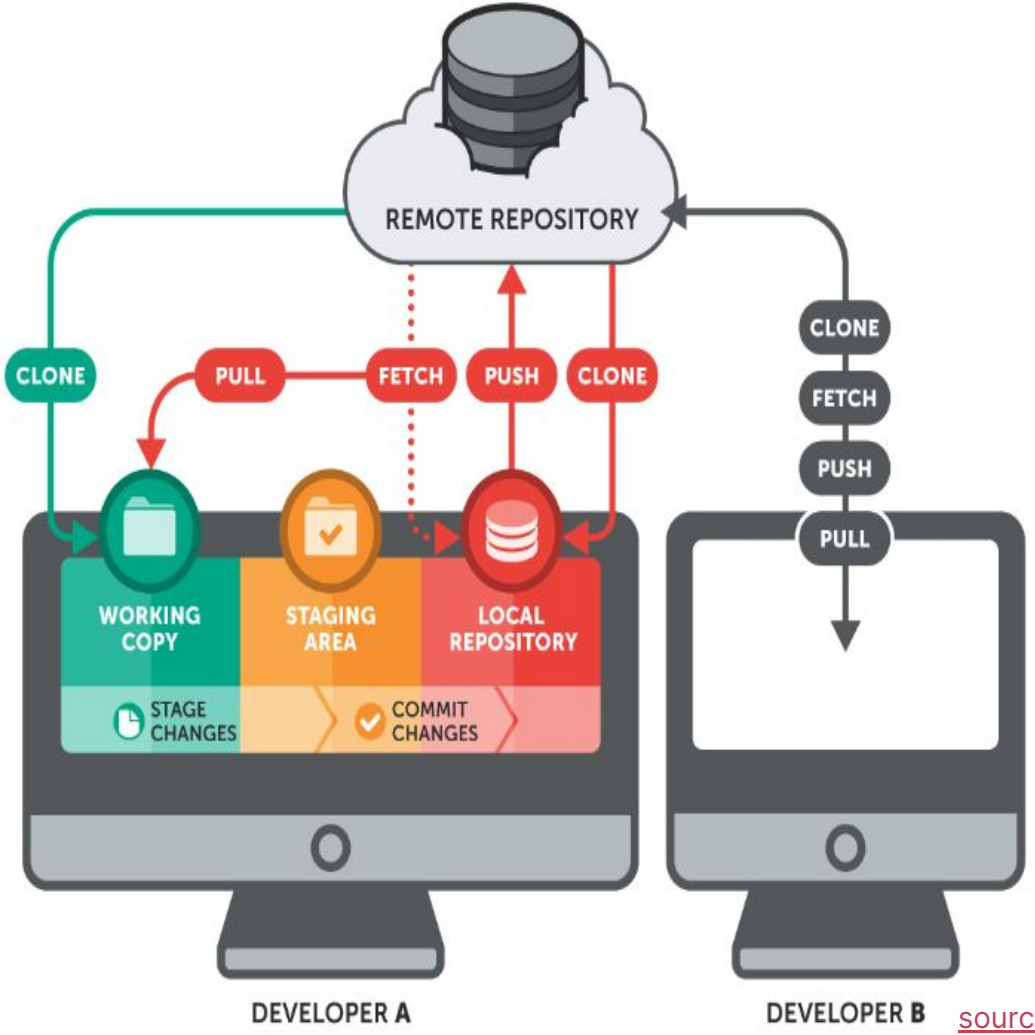
- TFS
- Perforce
- ClearCase
- Mercurial

Local vs Remote Repositories

/thoughtworks

Local vs Remote

- Local repositories reside on the computers of team members.
- Remote repositories are hosted on a server that is accessible for all team members - most likely on the internet or on a local network



source

Git vs GitHub

/thoughtworks

Git

- Git is a version control system
- Git is installed and maintained on your local system
- It is open source, free of cost
- Alternatives - TFS, SVN

GitHub

- GitHub is cloud based Hosting service
- Lets you share your code with others
- It is designed for profit, not free for enterprises
- Alternatives - GitLab, Azure DevOps, BitBucket

GUI Tools for Git

Git is a tool with no UI, We just have commands. Sometimes It's easier to have a UI for quick actions. There are tools available which provide some sort of UI for the GIT

- GitHub Desktop
- SourceTree
- List of third-party GUIs
- Jupyterlab git and github extensions
- GitToolBox plugin

To start learning it's encouraged to learn the commands without any UI. Learning the commands makes you more productive and efficient, not everything can be done on the GUI Tool. Some of the latest features/commands may not be available on the GUI tool.
Once you understand different commands, you may use these tools

How Git Works?

/thoughtworks

How Git Works

Git thinks of its data more like a series of snapshots of a miniature filesystem.

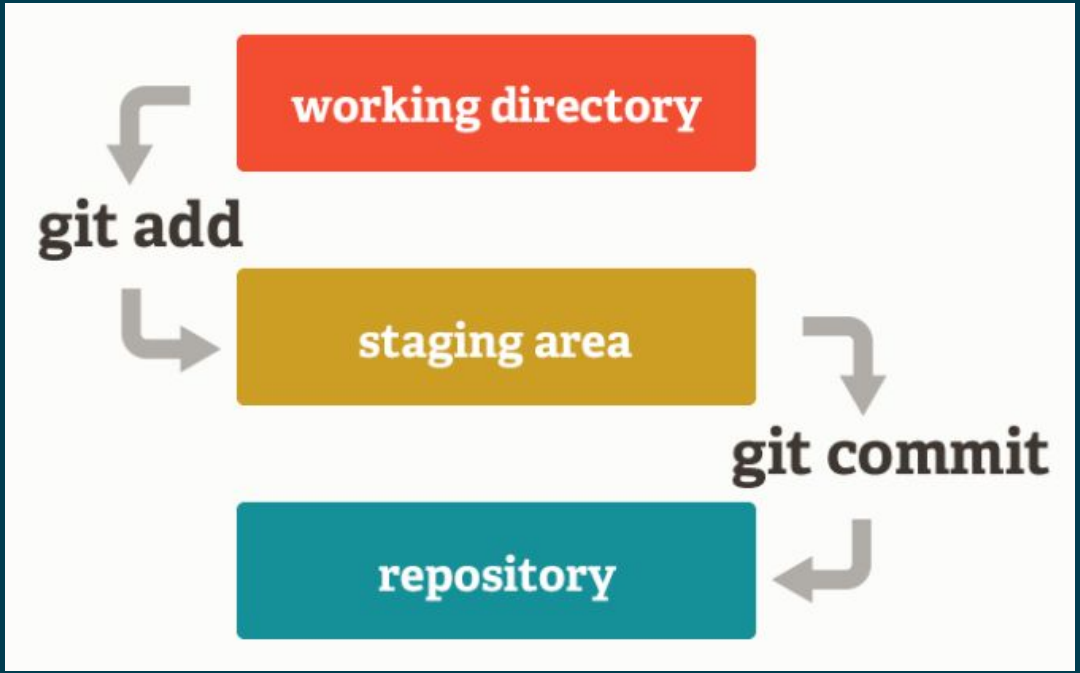
Every time you commit, or save the state of your project, Git basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot.

To be efficient, if files have not changed, Git doesn't store the file again, just a link to the previous identical file it has already stored.

- Git thinks about its data more like a stream of snapshots.
- Read more here

Three states in Git

/thoughtworks



/thoughtworks

How Git Works

The basic Git workflow goes something like this:

You modify files in your working tree.

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

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 Setup

/thoughtworks

Git Config

```
$ git version

$ git config --global user.name
<your_name>

$ git config --global user.email
<your_email>

$ git config --global --list
```

Git Repositories

New Git Repo

You can take a local directory that is currently not under version control, and turn it into a Git repository.

Go to your project folder

```
$ git init.
```

This creates a new subdirectory named .git — a Git repository skeleton

Existing Git Repo

You can clone an existing Git repository from elsewhere - internet or local network.

```
$ git clone <url>
```

Adding Remote Repositories

```
$ git remote add origin <url>
```

```
$ git remote -v
```

You may have multiple remote references

Ignoring file

.gitignore

Often, you'll have a class of files that you don't want to be tracked in Git.

These are generally automatically generated files such as log files or files produced by your build system. Here is an example .gitignore file:

```
# ignore all .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the TODO file in the current directory, not subdir/TODO
/TODO

# ignore all files in any directory named build
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt
```

Making Changes to a file

```
$ git version
```
(ensures git is installed)

```
$ git status

$ git add .

$ git status

$ git commit -m “Commit-message”

$ git push origin main
```

Modifying existing file

```
# Edit something

$ git status

$ git commit -m “Commit-message”

$ git push
```

Removing existing file

```
$ git rm <file_name>

$ git status

$ git commit -m “Commit-message”

$ git push
```

Git Stash

What happens when you are in middle of your task and suddenly Production issue comes up.

- You urgently need to stop working on your current task and work on fixing the bug
- Git provides a stash area, where you can save your work in progress and provides flexibility to switch to new branch/task

Saving your WIp in stash

```
$ git stash
```

The git stash command takes your uncommitted changes (both staged and unstaged), saves them away for later use, and then reverts them from your working copy

```
$ git stash pop
```

You can reapply previously stashed changes with

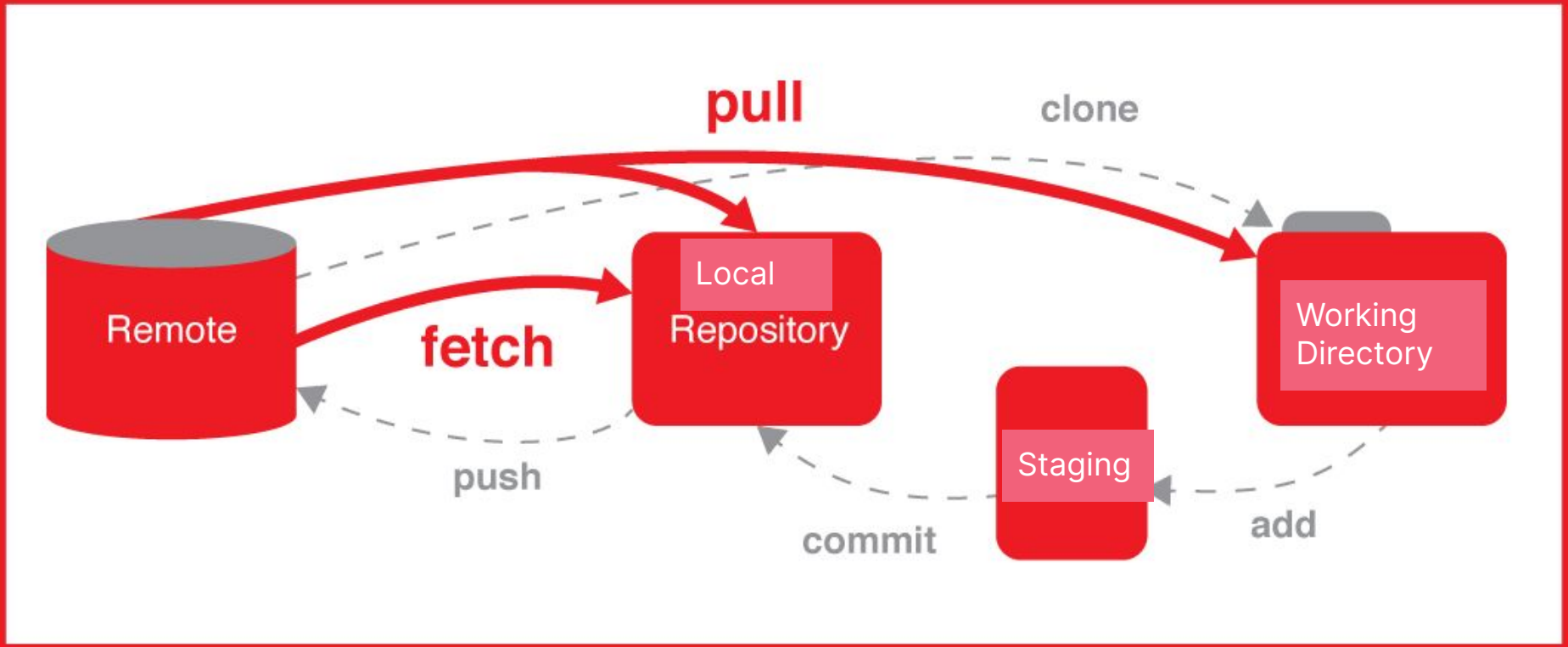
Read More

Getting Help

```
$ git help <command>
```

Fetch vs Pull

/thoughtworks



Fetch

- Git Fetch is the command that tells the local repository that there are changes available in the remote repository without bringing the changes into the local repository.

Pull

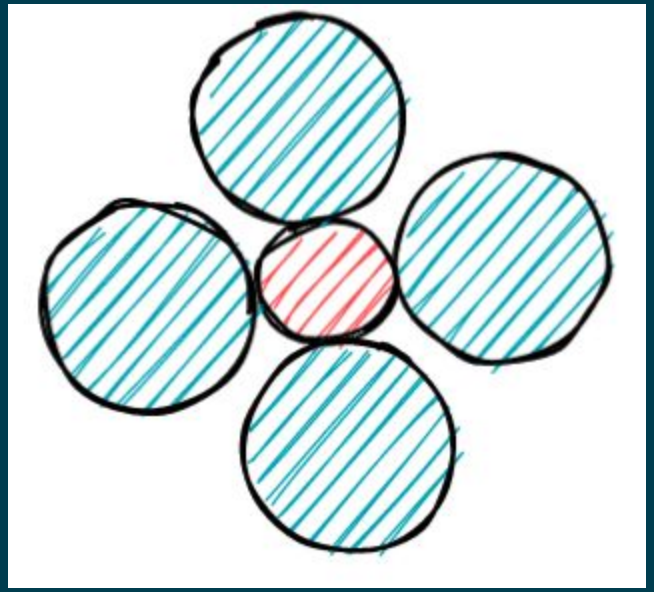
- Git Pull on the other hand brings the copy of the remote directory changes into the local repository. It applies all the changes to your local Repository

Key Points

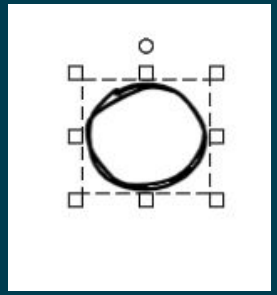
- **Status:** Show the status of the files, which files were modified, deleted or added.
- **Clone**: copying the remote repository to your local system
- **Add or stage**: taking changes you have made and get them ready to add to your git history
- **Commit**: add new or changed files to the git history for the repository
- **Pull**: get new changes others have added to the repository into your local repository
- **Fetch**: tells the local repository that there are changes available in the remote repository without bringing the changes into the local repository.
- **Push**: get changes from your local system onto the remote repository

Understanding Commits

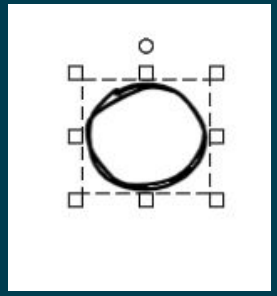
/thoughtworks



/thoughtworks

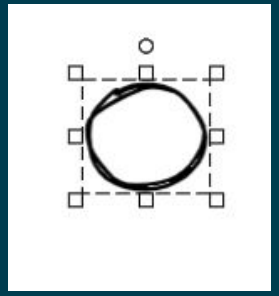


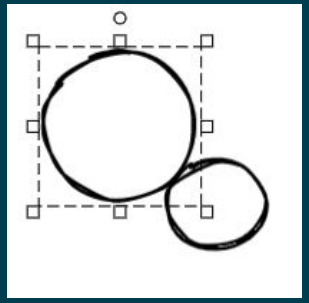
/thoughtworks



Commit 1
Id: 1
Date: 10the Sept 2022
Author: Rishita
Message: “Add center circle”

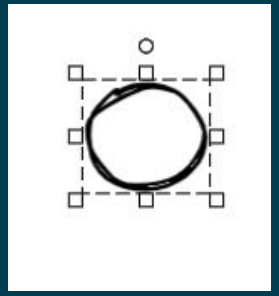
/thoughtworks



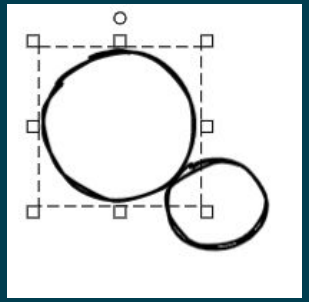


Commit 1
Id: 1
Date: 10the Sept 2022
Author: Rishita
Message: “Add center circle”

/thoughtworks

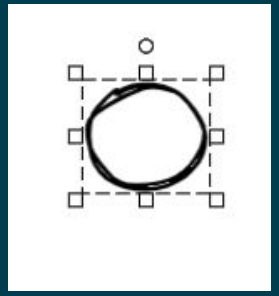


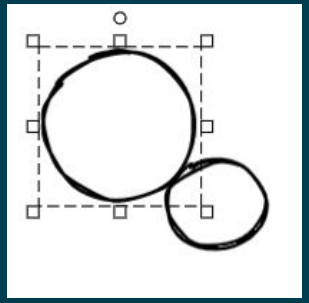
Commit 1
Id: 1
Date: 10the Sept 2022
Author: Rishita
Message: “Add center circle”

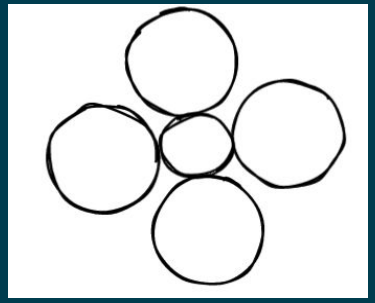


Commit 2
Id: 2
Date: 10the Sept 2022
Author: Rishita
Message: “Add Petal 1”

/thoughtworks



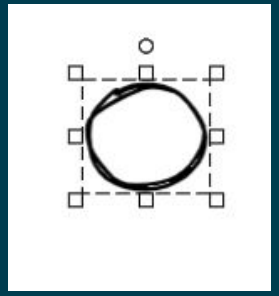




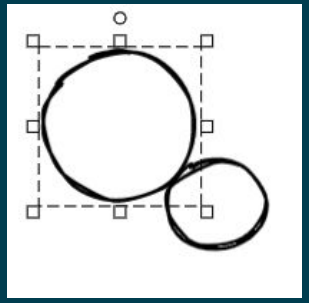
Commit 1
Id: 1
Date: 10the Sept 2022
Author: Rishita
Message: “Add center circle”

Commit 2
Id: 2
Date: 10the Sept 2022
Author: Rishita
Message: “Add Petal 1”

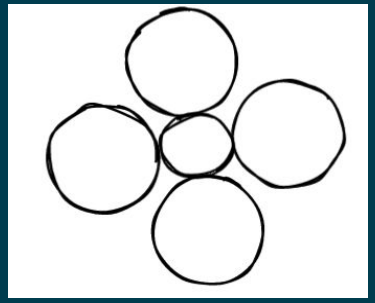
/thoughtworks



Commit 1
Id: 1
Date: 10the Sept 2022
Author: Rishita
Message: “Add center circle”

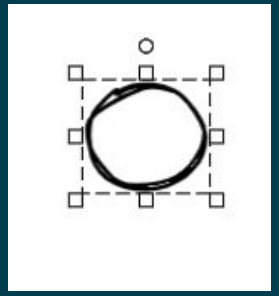


Commit 2
Id: 2
Date: 10the Sept 2022
Author: Rishita
Message: “Add Petal 1”

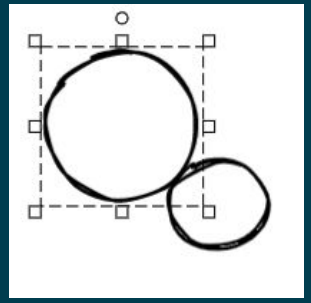


Commit 3
Id: 3
Date: 10the Sept 2022
Author: Rishita
Message: “Add Petal 2,3 ,4”

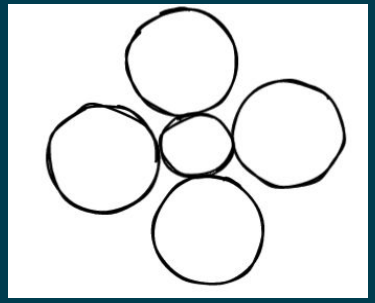
/thoughtworks



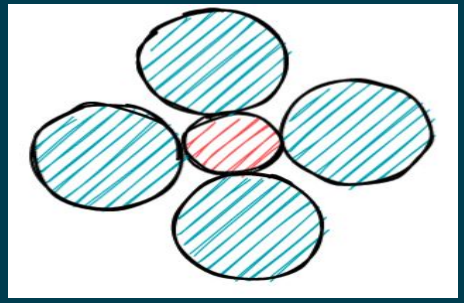
Commit 1
Id: 1
Date: 10the Sept 2022
Author: Rishita
Message: “Add center circle”



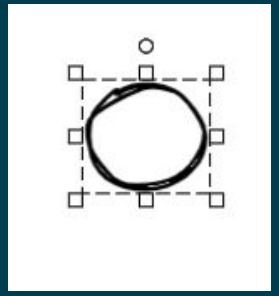
Commit 2
Id: 2
Date: 10the Sept 2022
Author: Rishita
Message: “Add Petal 1”



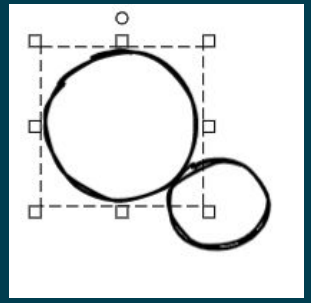
Commit 3
Id: 3
Date: 10the Sept 2022
Author: Rishita
Message: “Add Petal 2,3 ,4”



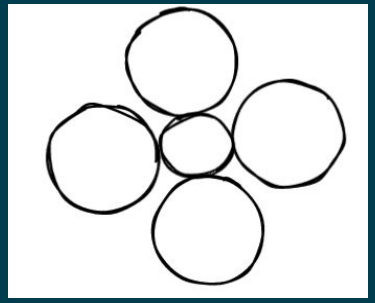
/thoughtworks



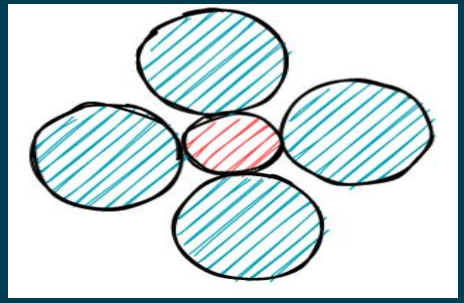
Commit 1
Id: 1
Date: 10the Sept 2022
Author: Rishita
Message: “Add center circle”



Commit 2
Id: 2
Date: 10the Sept 2022
Author: Rishita
Message: “Add Petal 1”

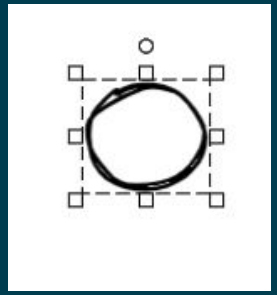


Commit 3
Id: 3
Date: 10the Sept 2022
Author: Rishita
Message: “Add Petal 2,3 ,4”

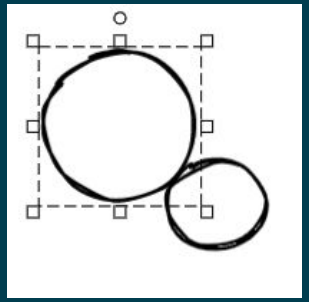


Commit 4
Id: 4
Date: 10the Sept 2022
Author: Rishita
Message: “Fill Color”

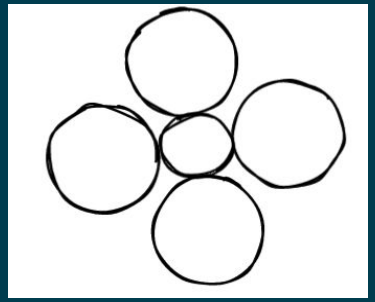
/thoughtworks



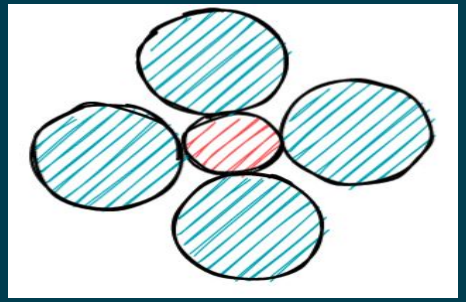
Commit 1
Id: 1
Date: 10the Sept 2022
Author: Rishita
Message: “Add center circle”



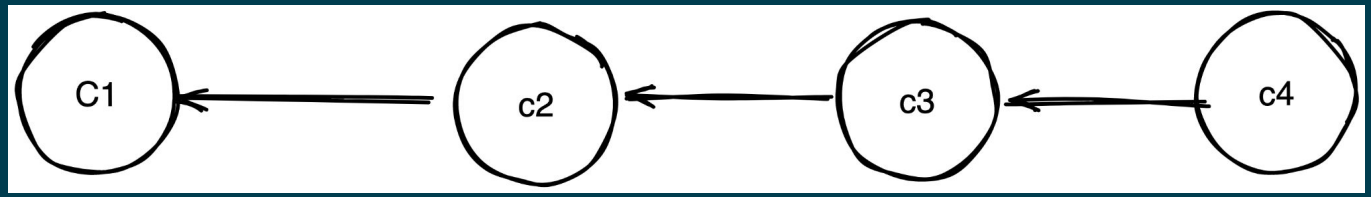
Commit 2
Id: 2
Date: 10the Sept 2022
Author: Rishita
Message: “Add Petal 1”



Commit 3
Id: 3
Date: 10the Sept 2022
Author: Rishita
Message: “Add Petal 2,3 ,4”



Commit 4
Id: 4
Date: 10the Sept 2022
Author: Rishita
Message: “Fill Color”

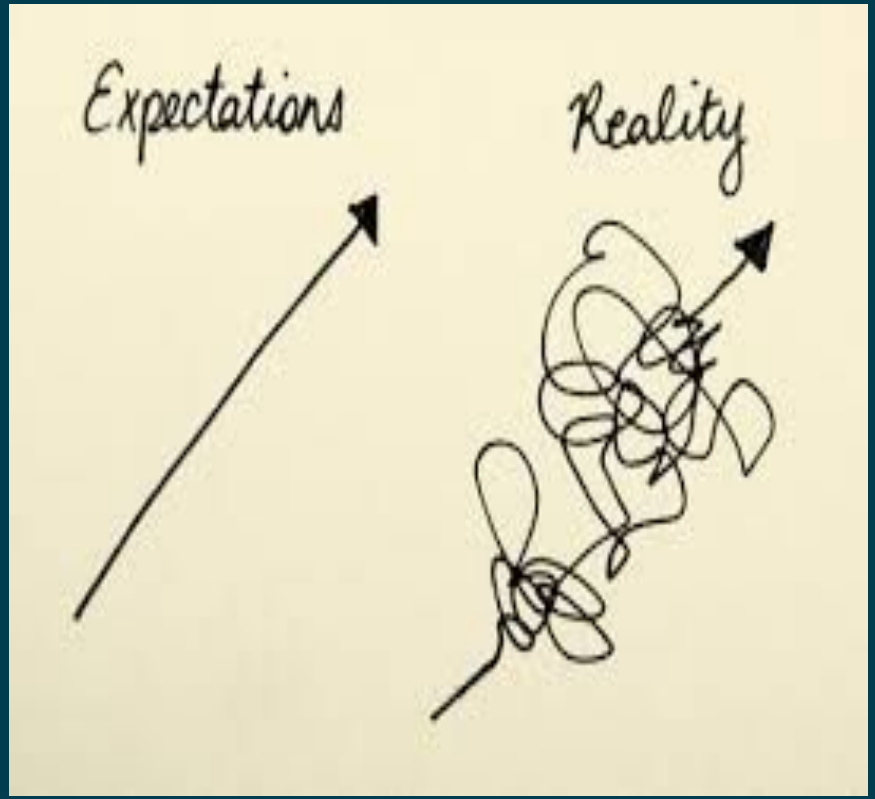


/thoughtworks

Viewing the Commit History

```
$ git log --pretty=oneline

$ git log --pretty=format:"%h - %an, %ar : %s"

$ git log --graph --abbrev-commit --decorate

--first-parent <branch_name>

$ git log --since=2.weeks
```

Undoing Things



/thoughtworks

Amend Commit

/thoughtworks

Amend Commit - Undo The last commit in local

Effectively, it’s as if the previous commit never happened, and it won’t show up in your repository history.

The obvious value to amending commits is to make minor improvements to your last commit.

Avoids cluttering your repository history with commit messages of the form, “Oops, forgot to add a file” or “fixing a typo in last commit”.

- To add some missing files
- To change your commit message.
- To make the additional changes you forgot
- Only amend commits that are still local and have not been pushed to remote.

.

Amend

```
$ git commit -m 'Initial commit'

$ git add forgotten_file

$ git commit --amend
```

You end up with a single commit — the second commit replaces the results of the first.

Reset

/thoughtworks

Reset - Undo commits in local

git reset can be invoked with the --hard or --soft options

-hard

Resets the index and working tree. Any changes to tracked files in the working tree since `<commit>` are discarded. Any untracked files or directories in the way of writing any tracked files are simply deleted.

-soft

Does not touch the index file or the working tree at all (but resets the head to `<commit>`, just like all modes do). This leaves all your changed files "Changes to be committed", as `git status` would put it.

- Can be applied to any commit not just last commit
- Good practice is to reset commits that are still local and have not been pushed to remote.
- Explore more options- https://www.atlassian.com/git/tutorials/undoing-changes/git-reset https://git-scm.com/docs/git-reset

Reset - Undo commits in local

--hard

The Commit History ref pointers are updated to the specific commit.

Then, the Staging Index and Working Directory are reset to match that of the specific commit.

Any previously pending changes to the Staging Index and the Working Directory gets reset to match the state of the Commit Tree.

- This means any pending work that was hanging out in the Staging Index and Working Directory will be lost.

Reset - Undo commits in local

--soft

When the --soft argument is passed, the ref pointers are updated and the reset stops there. The Staging Index and the Working Directory are left untouched.

- Your work in is not lost.

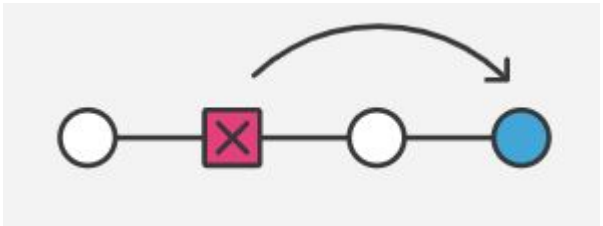
Revert

/thoughtworks

Revert

The git revert command instead of removing the commit from the project history, it figures out how to invert the changes introduced by the commit and appends a new commit with the resulting inverse content.

- To undo a public commit
- To make the additional changes you forgot
- Unlike reset it prevents Git from losing history.

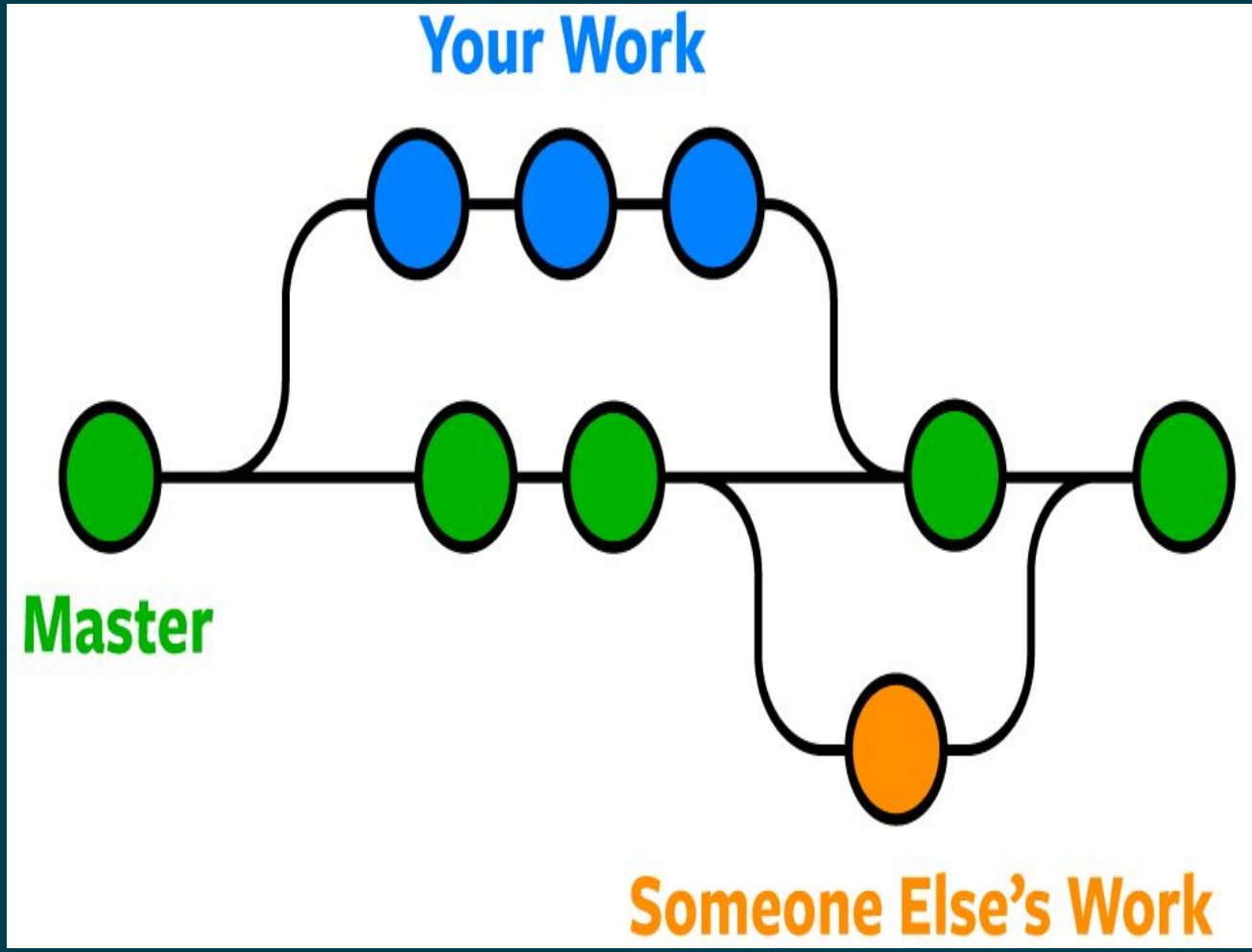


. source

Key Points

- Once changes have been committed they are generally permanent
- Use Amend commit to fix last commit in your local
- git revert is the best tool for undoing shared public changes
- git reset is best used for undoing local private changes

Branching



/thoughtworks

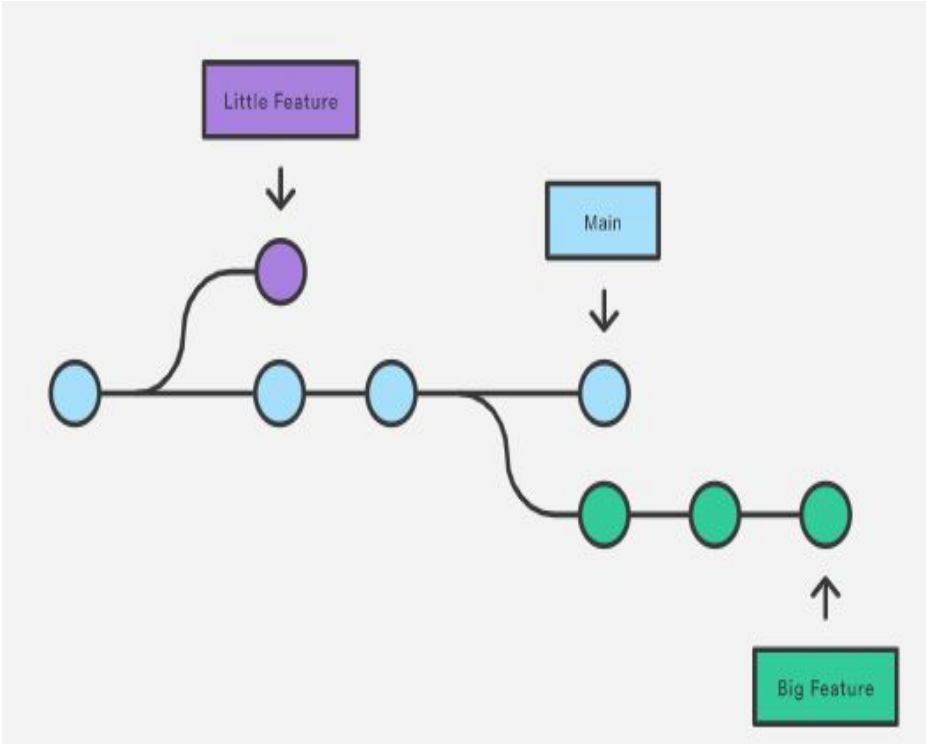
Why Branching

/thoughtworks

Why Branching

You are working on some feature but suddenly a Prod issue comes up which needs an urgent fix

You want to switch between two features which are independent of each other



Branching

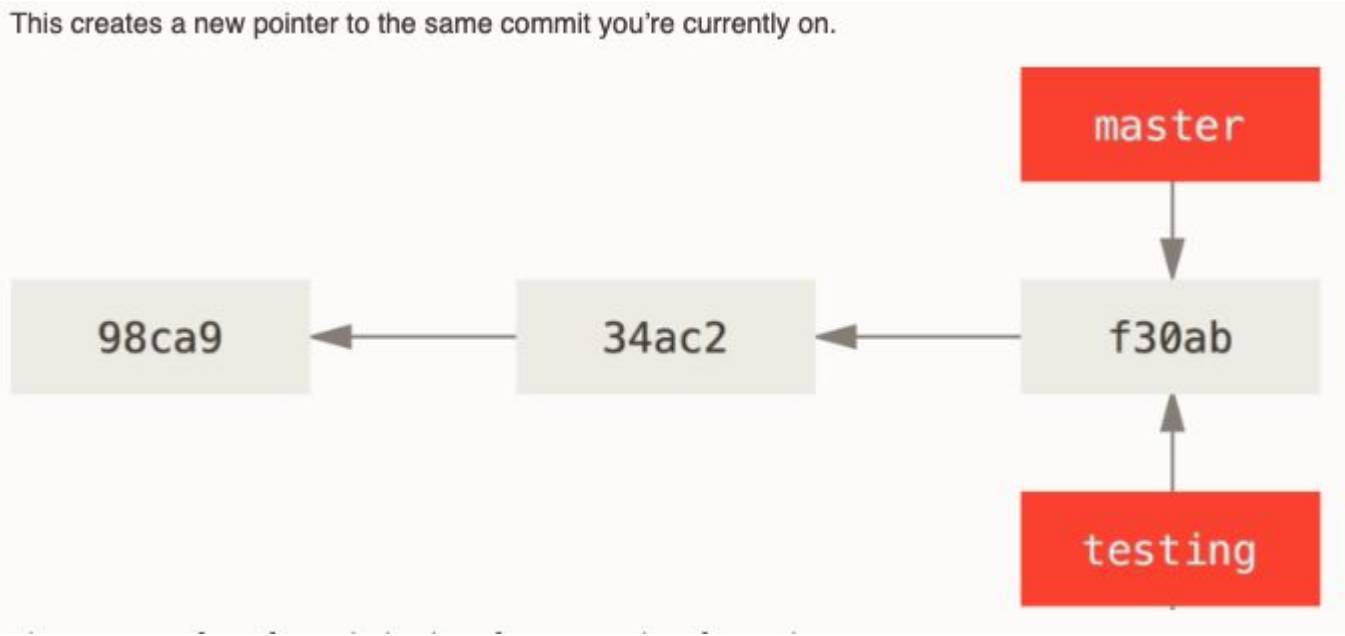
A branch in Git is simply a lightweight movable pointer to one of these commits.

As you start making commits, you're given a main branch that points to the last commit you made.

Every time you commit, the main branch pointer moves forward automatically.

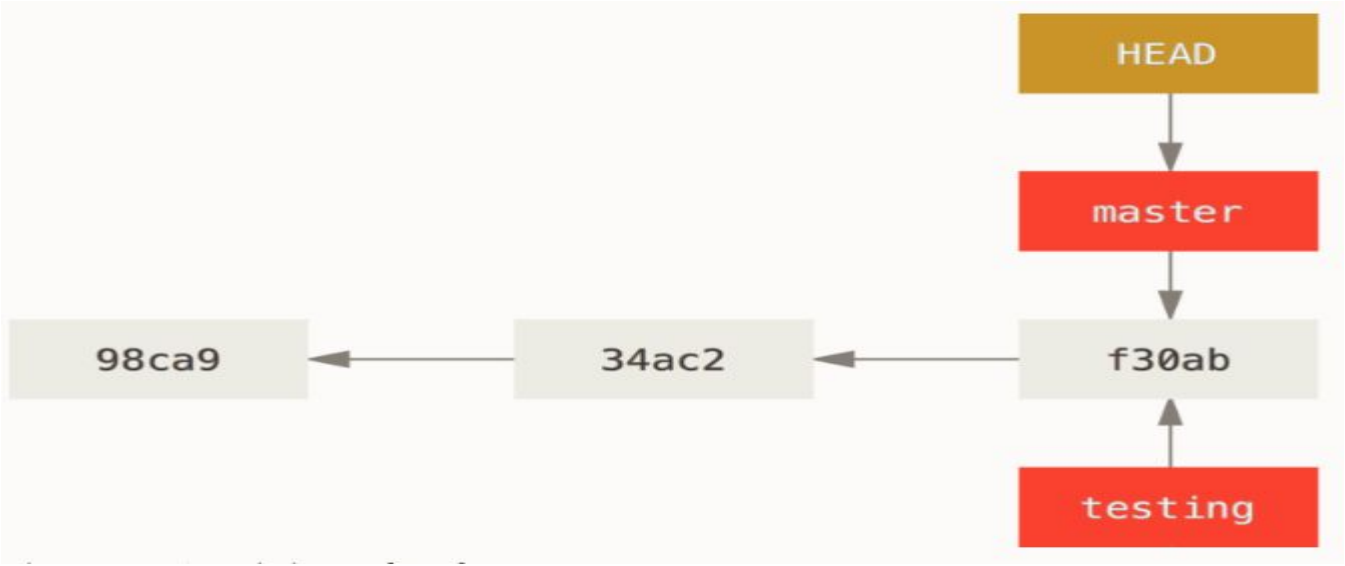
Creating a New Branch

```
$ git branch testing
```



You are still on Master

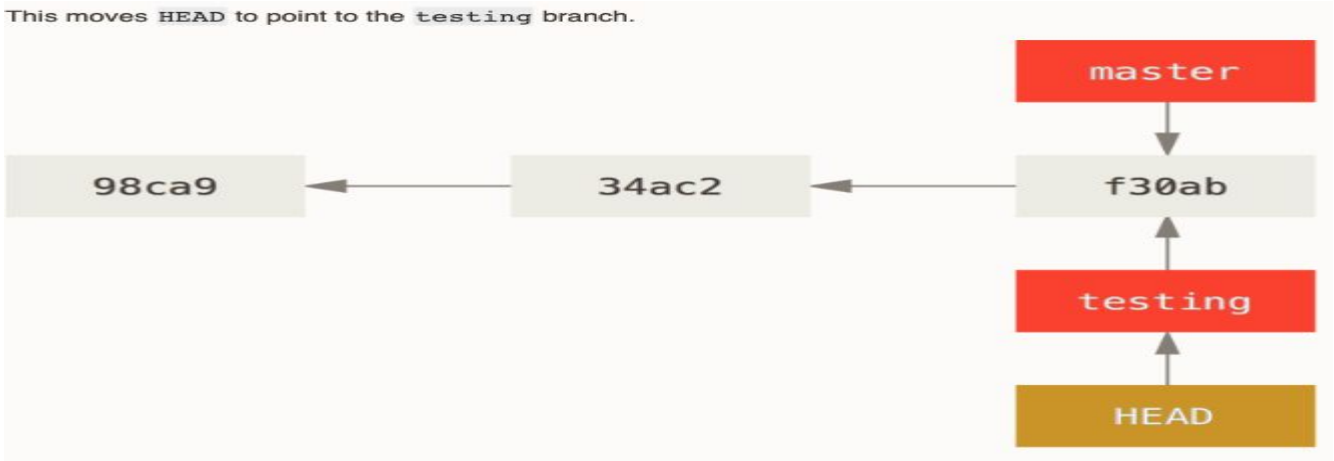
```
$ git status
```



Switching to New Branch

```
$ git checkout testing
```

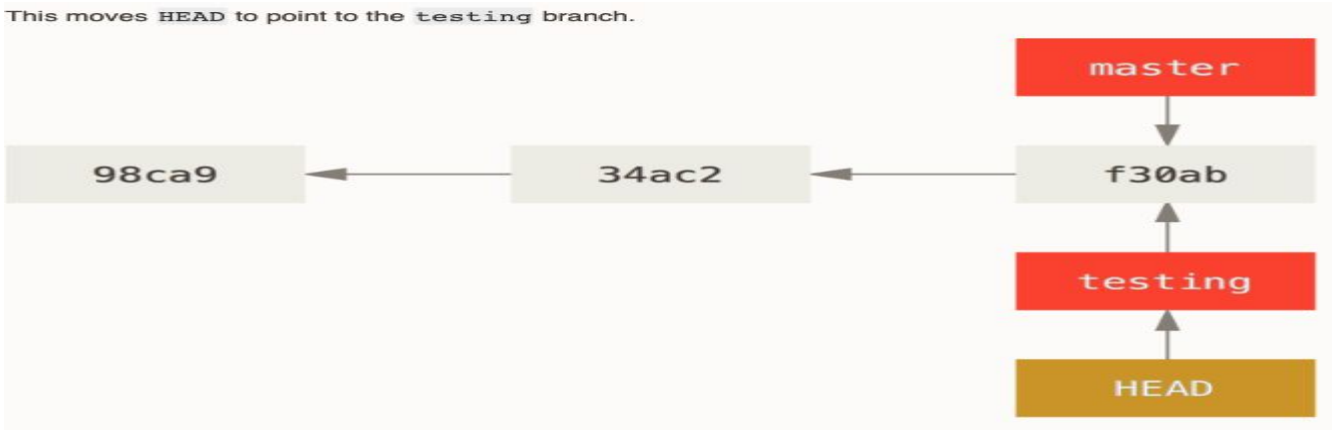
```
$ git switch testing
```



Create and switch to New Branch in one command

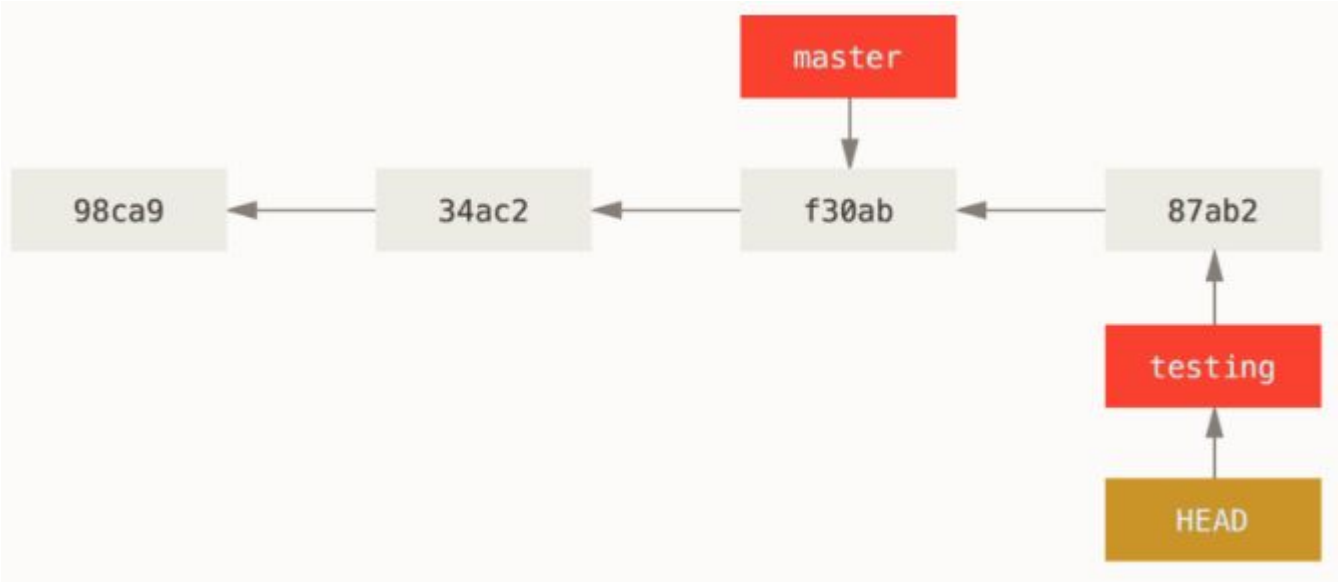
```
$ git checkout -b testing
```

```
$ git switch -c testing
```
(-c for create flag)



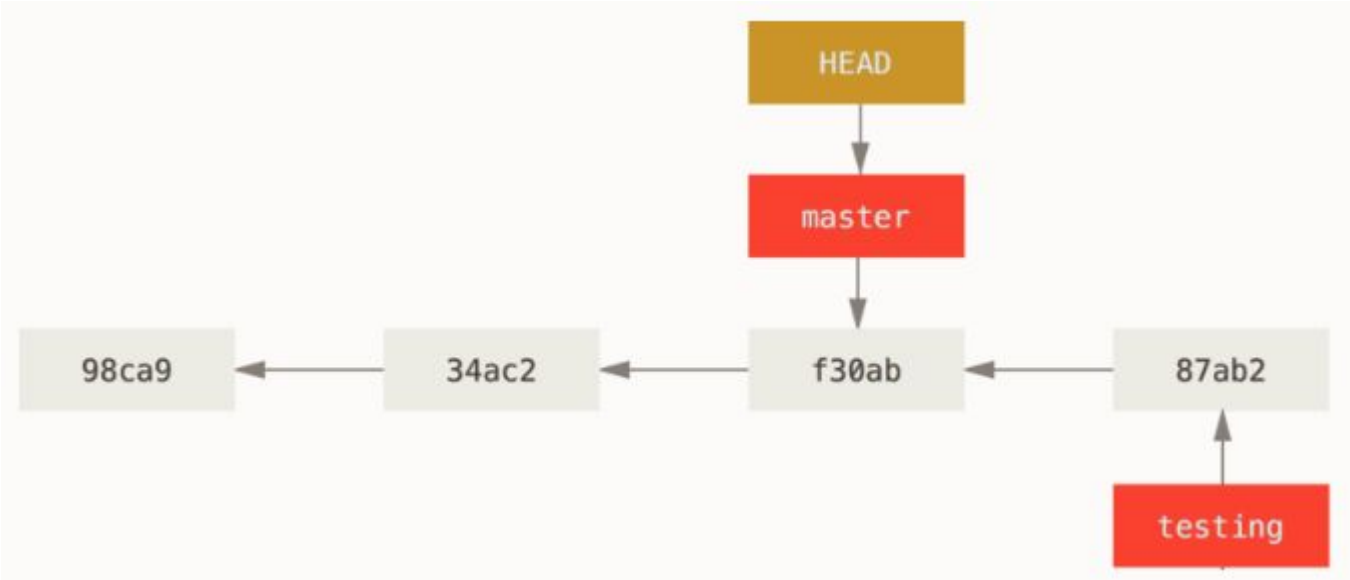
Commit on New Branch

```
$ git commit -a -m “made some changes”
```



Switching back to Master

```
$ git checkout master
```



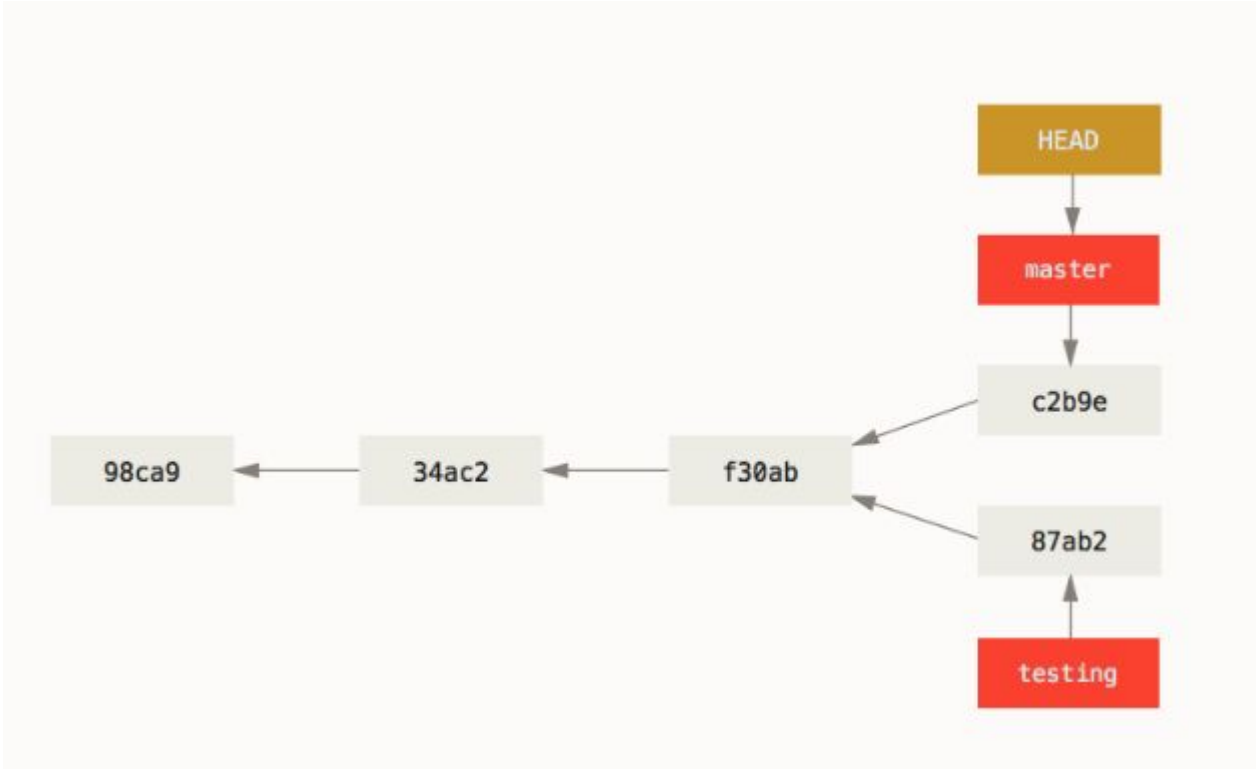
Switching back to Master

Switching branches changes files in your working directory

If you switch to an older branch, your working directory will be reverted to look like it did the last time you committed on that branch.

If Git cannot do it cleanly, it will not let you switch at all.

Master and testing branch can progress independently



Deleting existing Branch

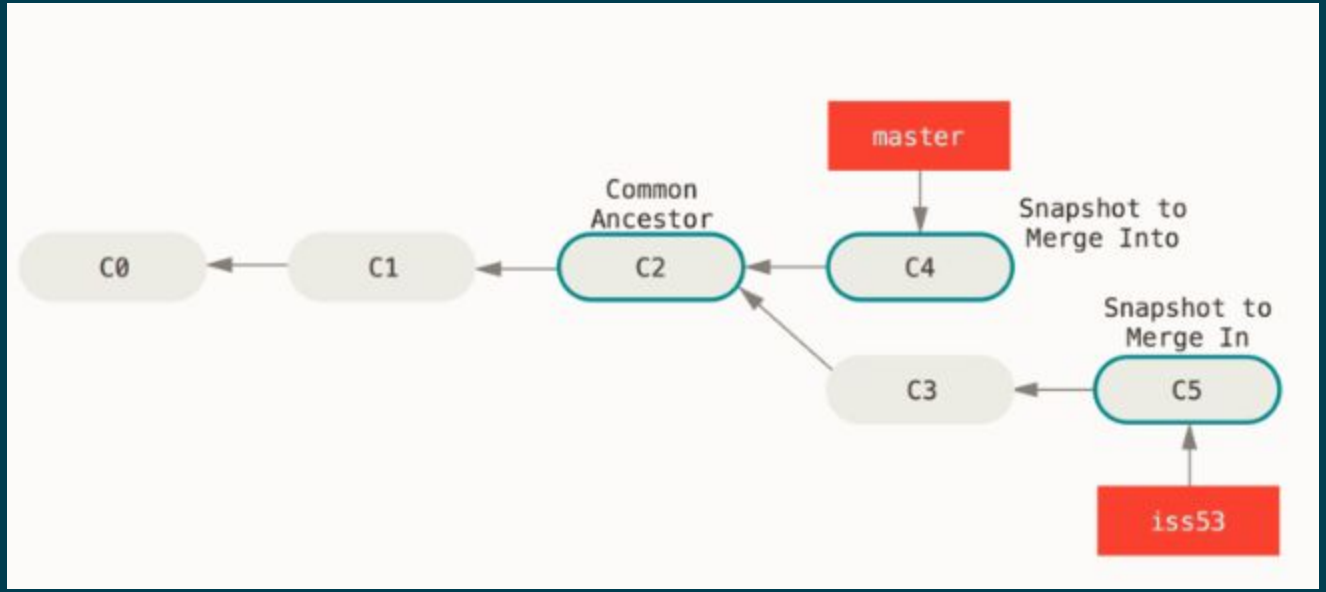
```
$ git branch -d <branch_name>
```

The branch may still exist at remote repository

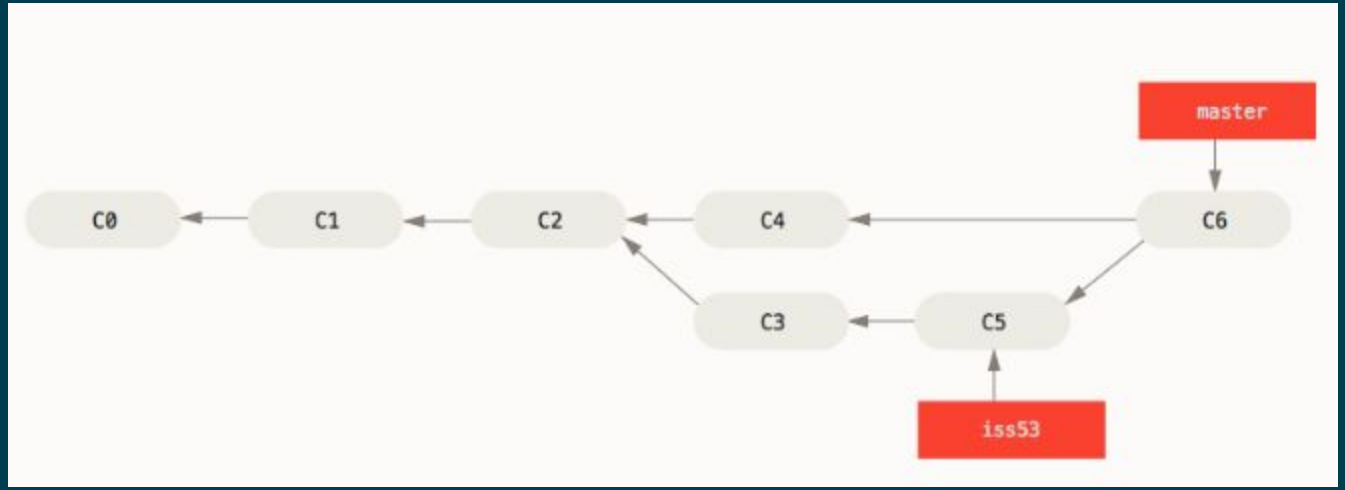
```
$ git push origin -d <branch_name>
```

Merging

/thoughtworks



/thoughtworks



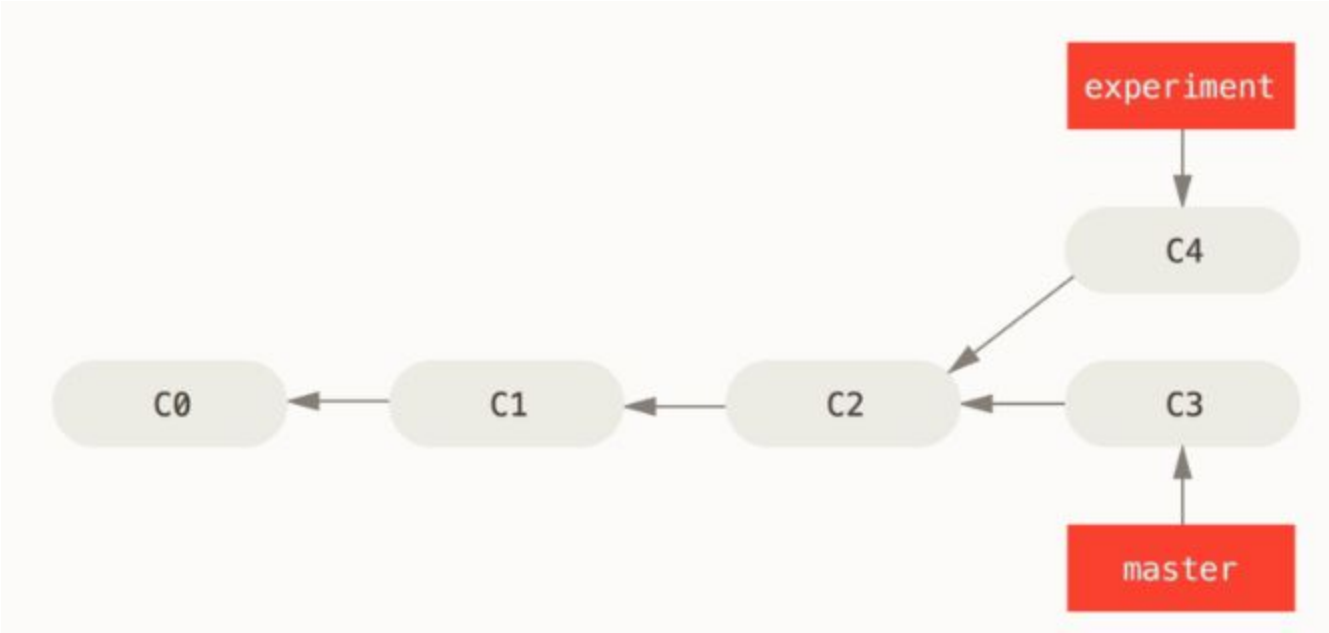
/thoughtworks

Branch commands

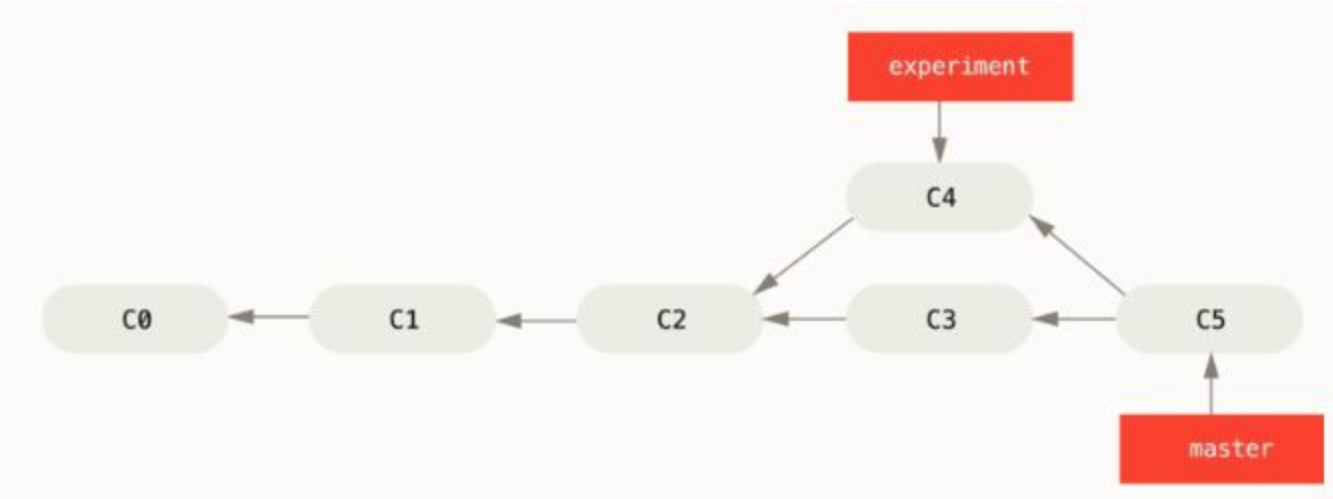
```
$ git merge testing

$ git branch

$ git branch -v

$ git branch --merged

$ git branch --no-merged

$ git branch --move <bad-branch-name>
<corrected-branch-name>
```

Rebase

/thoughtworks

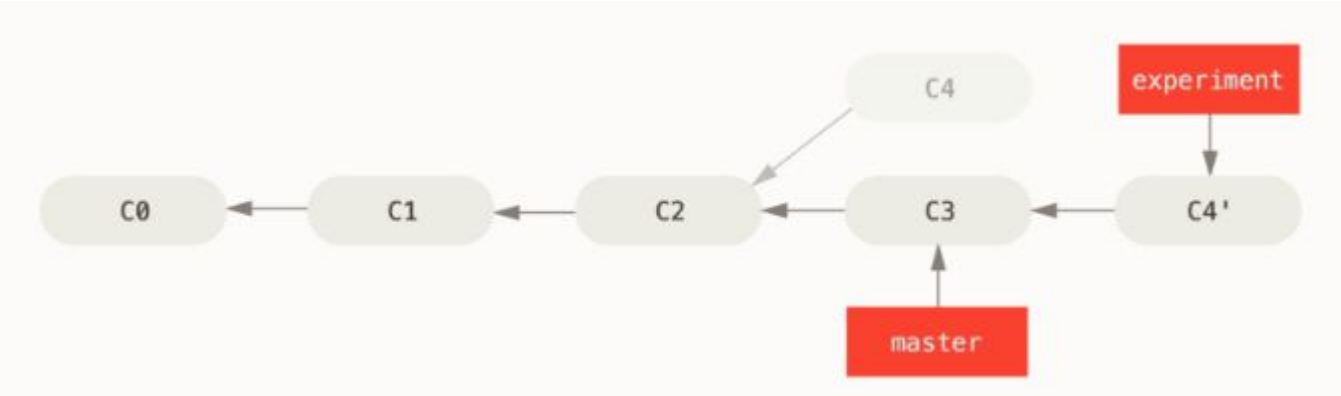


Merge experiment into master branch



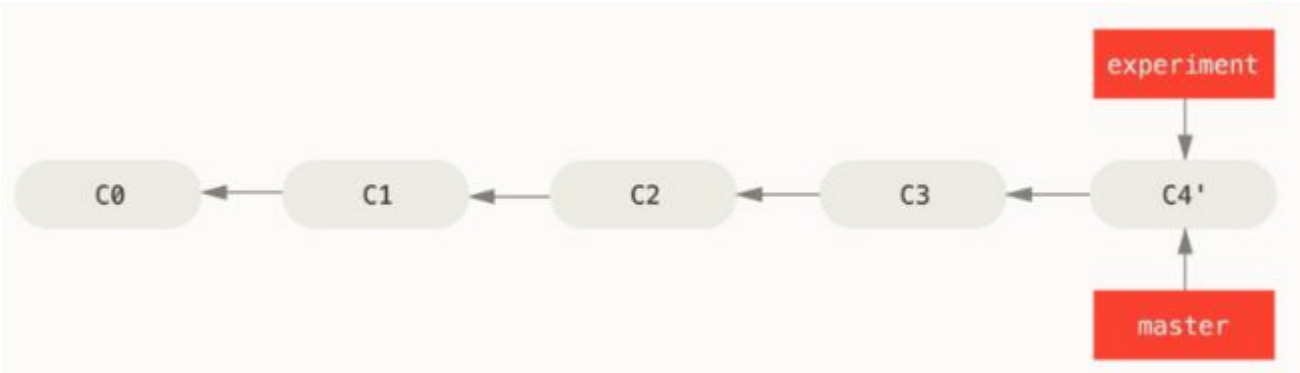
Rebase master branch

```
$ git checkout experiment
$ git rebase master
```



Fast Forwarding master Branch

```
$ git checkout master
$ git merge experiment
```



Good Practice

```
$ git pull --rebase
```

Before pushing your changes, it’s a good practice to take all the changes that others have already done and put your changes on top of that.

Resolving conflicts

```
$ git diff
```

Trunk-Based Development

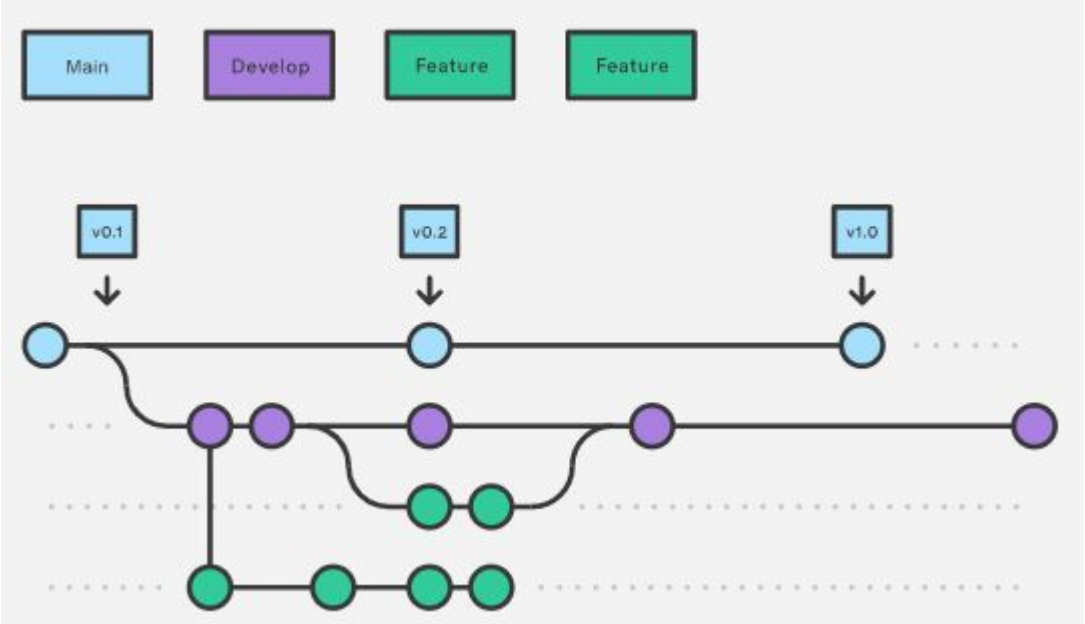
Trunk-based development is a version control management practice where developers merge small, frequent updates to a core “trunk” or main branch.

- Develop in small batches
- Linear history of commits
- Feature flags
- Implement comprehensive automated testing
- Perform asynchronous code reviews
- Have three or fewer active branches in the application's code repository
- Merge branches to the trunk at least once a day
- Build fast and execute immediately

Feature Branch Workflow

Each new feature should reside in its own branch

When you're done with the development work on the feature, the next step is to merge the feature_branch into develop.



Resources

Always read at source

- https://git-scm.com/doc

Udemy Course

- Git & GitHub - The Practical Guide by Maximilian Schwarzmüller

Doc tutorial

- https://www.atlassian.com/git/tutorials/setting-up-a-repository

Git Cheat-Sheet

- Atlassian
- https://ndpsoftware.com/git-cheatsheet.html#loc=index;
- freeCodeCamp

Picture

- https://girliemac.com/blog/2017/12/26/git-purr/

Q & A

/thoughtworks