CHITKARA UNIVERSITY

Dr. XYZ
Asst. Professor, DCSE,
Chitkara University, Punjab

"Source Code Management – CS181"

G-02/05

Offered by:

"Department of Computer Science and Engineering,
Chitkara University Institute of Engineering and Technology, Punjab"

CHITKARA UNIVERSITY

Module 1st

Git and Version Control

CHITKARA UNIVERSITY

Agenda

Objective - Tools for advanced programming management

We aim to **"Explore the Potential"** of open source code software management to provide all a better and more useful platform

01 Functionality Examination
Examine the functionality of Software Version Control Systems.

02 Functionality Utilization
Utilize the functionality of GIT to support version control of source code.

03 Workflow Analysis
Assess workflows in various version control systems like Git.

04 Workflow Control
Apply the workflows to create collaboration with Co Participants

Timeline

CHITKARA UNIVERSITY

Session 1st

Version Control – Git client (CLI, GUI), Installing git CLI and Git GUI, Initializing the repository, and exploring git --help
Exploring Github – Creating repository, controls on the panel, working on Git Hub alone

Session 2nd

Working With Git – Commands for initiating repos, managing repos, Git status, add, commit, stage – Life cycle of a file in Git managed in Repos, Git branches and HEAD, Git branches management

Session 3rd

Git Cloning - Cloning repository, Exploring contents of the cloned repository, Exploring cloned repository in GitHub Desktop, Commit changes in the cloned repository, Git Configuration Files, attributes for managing, filtering, masking

Session 4th

Working With Git History – Forensics on GIT logs
Log, undo changes in history – creating presentable GUI for GIT activity in versioned repos
Merge Resolution In Git – Branching, tagging branches, creating test

Session 5th

Git branch, merge resolution workflow- Resolution of merge conflicts, GitHub - Cloning remote repository, Git push, fetch, and pull operations - Pushing to the remote repository, Git pull with fast forward merge, Resolving conflicts during Git pull

CHITKARA UNIVERSITY

Session 1st

Git and Version Control

- Introducing Version Control
- Git client (CLI, GUI)
- Linux environment Emulation
- Installing Git CLI and git GUI
- Initializing the repository,
- Exploring git --help
- Exploring Github and
- Creating a Public Repository
- Understanding controls on the panel
- Working on Git Hub alone
- Realizing the significance of Git Client for Github utilization

CHITKARA UNIVERSITY

KEEP TRACK MODIFICATION WITH

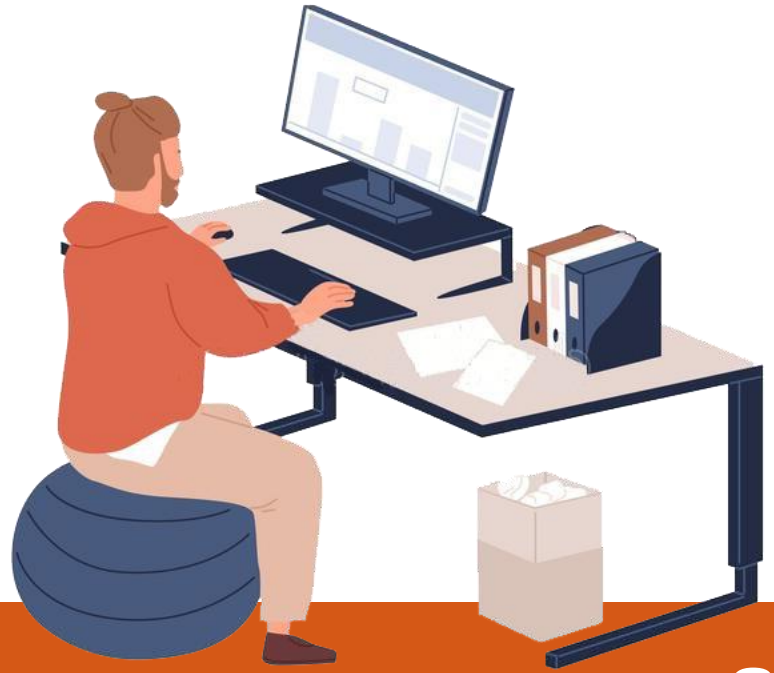
Version Control System

Version control systems are software tools that help software teams manage changes to source code over time. As development environments have accelerated, version control systems help software teams work faster and smarter.

USE CASE	INTRODUCTION	BENIFITS	BEST VERSION(s)

Use Case (Version Control)

CHITKARA UNIVERSITY



Let's say there's a multinational company, that has offices and employees all around the globe...

Set of Challenges Company Faces

Collaboration

Storing Versions

Restoring Previous Version

Figuring out the reason behind

Backup

CHITKARA UNIVERSITY

POSSIBLE SOLUTION

Version Control System

A version Control system records all the changes made to a file or set of files, so a specific version may be called later if needed.

How version control helps high performing development and DevOps teams prosper

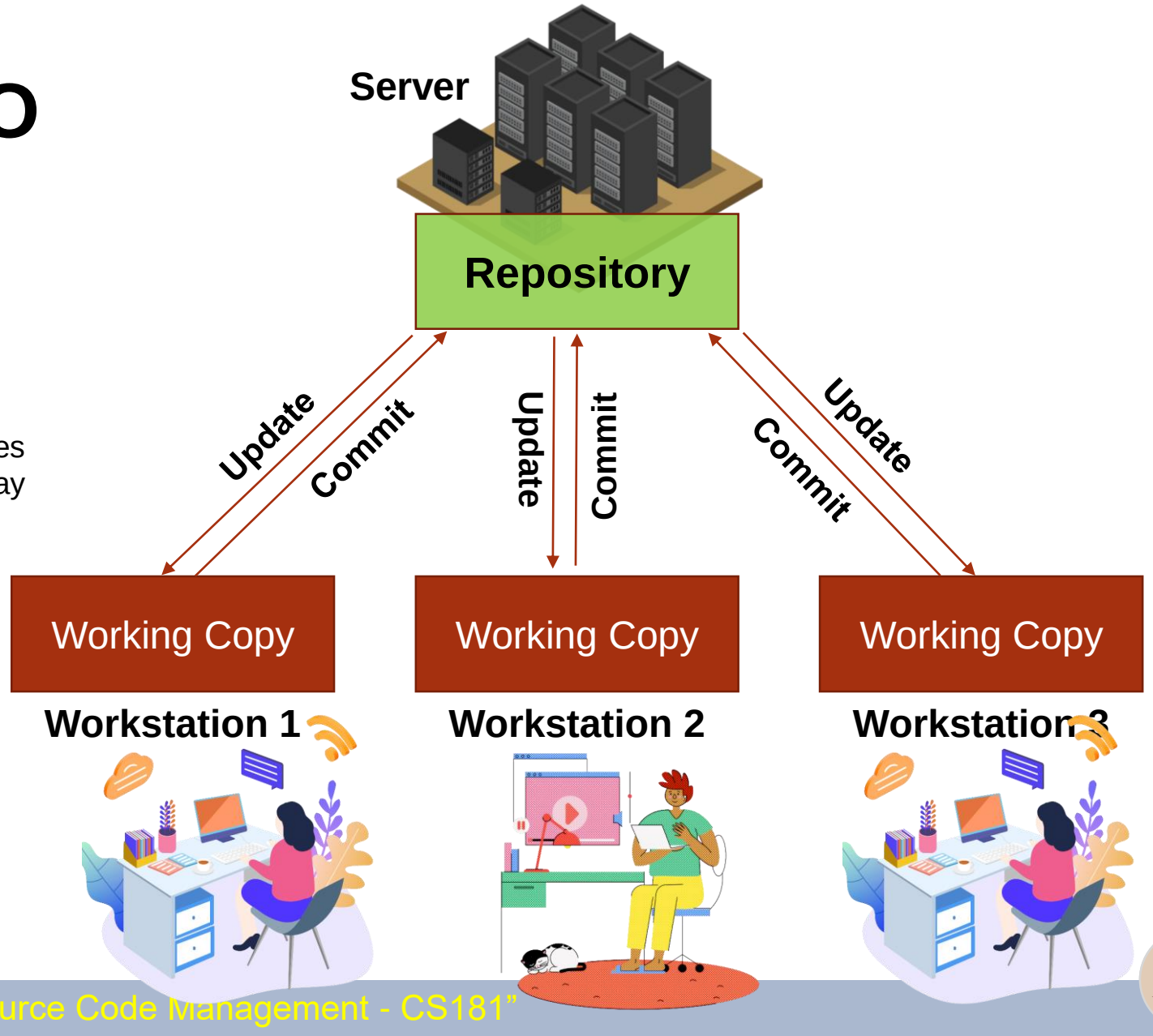
- Version control, also known as source control.
- Is the practice of tracking and managing changes to software code.
- Version control systems are software tools that help software teams manage changes to source code over time.
- As development environments have accelerated, version control systems help software teams work faster and smarter.
- They are especially useful for DevOps teams since they help them to reduce development time and increase successful deployments.
- Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.
- Version control protects source code from both catastrophe and the casual degradation of human error and unintended consequences.
- Version control helps teams solve these kinds of problems, tracking every individual change by each contributor and helping prevent concurrent work from conflicting.

CHITKARA UNIVERSITY

WORKING SENARIO

Version Control System

A version Control system records all the changes made to a file or set of files, so a specific version may be called later if needed.

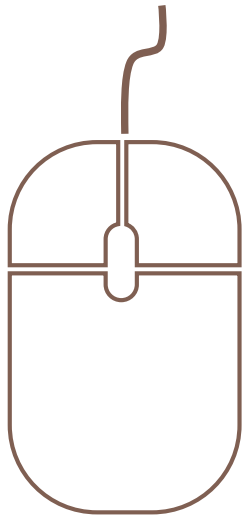


Benefits of (Version Control)

CHITKARA UNIVERSITY

Using version control software is a best practice for high performing software and DevOps teams. Version control also helps developers move faster and allows software teams to preserve efficiency and agility as the team scales to include more developers.

- ❖ Version Control Systems (VCS) have seen great improvements over the past few decades and some are better than others.
- ❖ VCS are sometimes known as SCM (Source Code Management) tools or RCS (Revision Control System).
- ❖ One of the most popular VCS tools in use today is called Git. Git is a Distributed VCS, a category known as DVCS, more on that later. Like many of the most popular VCS systems available today, Git is free and open source.
- ❖ Regardless of what they are called, or which system is used, the primary benefits you should expect from version control are as follows.



VCS Benefits

Branching and Merging

Creating a "branch" in VCS tools keeps multiple streams of work independent from each other while also providing the facility to merge that work back together, enabling developers to verify that the changes on each branch do not conflict.

Traceability

Being able to trace each change made to the software and connect it to project management and bug tracking software such as Jira, and being able to annotate each change with a message describing the purpose and intent of the change can help not only with root cause analysis and other forensics.

Version Comparison

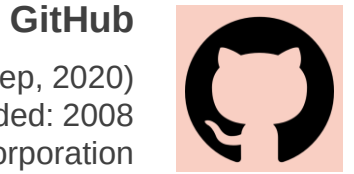
A complete long-term change history of every file. This means every change made by many individuals over the years. Changes include the creation and deletion of files as well as edits to their contents. Different VCS tools differ on how well they handle renaming and moving of files.

Supports Developer's Workflow

Having the annotated history of the code at your fingertips when you are reading the code, trying to understand what it is doing and why it is so designed can enable developers to make correct and harmonious changes that are in accord with the intended long-term design of the system.

Best (Version Control Systems)

CHITKARA UNIVERSITY



Bitbucket

Date launched: 2008
Owner: Atlassian

GitHub

Users: 56 million (as of Sep, 2020)
Founded: 2008
Parent organization: Microsoft Corporation

Apache Subversion

Initial release: 20 October 2000
Developer(s): Apache Software Foundation
License: Apache-2.0

Best VCS

GitLab

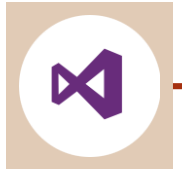
Owner: GitLab Inc. Nasdaq: GTLB
Launched: 2014

Perforce

Owner: Clearlake Capital, Francisco Partners
Founded: 1995

Team Foundation Server

Initial release: 2005
Developer(s): Microsoft
License: Trialware



Mercurial

Release date: 19 April 2005
License: GPL-2.0-or-later

AWS CodeCommit

Initial release: July 9, 2015
Hosted: Amazon Web Services

Concurrent Version control

Founded: 1986
Developer: FLOSS community

Beanstalk

Owner: Wildbit
Founded: 1999 in Philadelphia.

CHITKARA UNIVERSITY

All version control systems using a programming environment to provide the whole class of software solutions with

Source Code Management

Software Configuration Management (SCM)

- Generic term for the ability to manage multiple version of
  - a document
  - a collection of documents

INTRODUCTION | IMPORTANCE | BENIFITS | BEST PRACTICEs

CHITKARA UNIVERSITY

INTRODUCTION

SOURCE CODE MANAGEMENT

is an integral part of any development project in the current IT world.

How Source code management support storing the work results and sharing the details with colleagues is an essential part

- ❖Source code management (SCM) is used to track modifications to a source code repository.
- ❖SCM is also synonymous with Version control.
- ❖SCM tracks a running history of changes to a code base and helps resolve conflicts when merging updates from multiple contributors.
- ❖As software projects grow in lines of code and contributor head count, the costs of communication overhead and management complexity also grow.
- ❖SCM is a critical tool to alleviate the organizational strain of growing development costs.

Importance of

CHITKARA UNIVERSITY

Source Code Management Tools

- When multiple developers are working within a shared codebase it is a common occurrence to make edits to a shared piece of code.
- Separate developers may be working on a seemingly isolated feature, however this feature may use a shared code module.
- Therefore developer 1 working on Feature 1 could make some edits and find out later that Developer 2 working on Feature 2 has conflicting edits.

- Before the adoption of SCM this was a nightmare scenario. Developers would edit text files directly and move them around to remote locations using FTP or other protocols.
- Developer 1 would make edits and Developer 2 would unknowingly save over Developer 1's work and wipe out the changes.
- SCM's role as a protection mechanism against this specific scenario is known as Version Control.

- ✓ **SCM brought version control safeguards to prevent loss of work due to conflict overwriting.**
- ✓ **These safeguards work by tracking changes from each individual developer and identifying areas of conflict and preventing overwrites.**
- ✓ **SCM will then communicate these points of conflict back to the developers so that they can safely review and address.**

- This foundational conflict prevention mechanism has the side effect of providing passive communication for the development team.
- The team can then monitor and discuss the work in progress that the SCM is monitoring.
- The SCM tracks an entire history of changes to the code base.
- This allows developers to examine and review edits that may have introduced bugs or regressions.

CHITKARA UNIVERSITY

BENEFITS OF SOURCE CODE MANAGEMENT

In addition to version control SCM provides a suite of other helpful features to make collaborative code development a more user friendly experience. Once SCM has started tracking all the changes to a project over time, a detailed historical record of the projects life is created. This historical record can then be used to 'undo' changes to the codebase. The SCM can instantly revert the codebase back to a previous point in time. This is extremely valuable for preventing regressions on updates and undoing mistakes.

History log

01

- The SCM archive of every change over a project's life time provides valuable record keeping for a project's release version notes.
- A clean and maintained SCM history log can be used interchangeably as release notes.
- This offers insight and transparency into the progress of a project that can be shared with end users or non-development teams.

Non-overlapping

02

- SCM will reduce a team's communication overhead and increase release velocity.
- Without SCM development is slower because contributors have to take extra effort to plan a non-overlapping sequence of develop for release.
- With SCM developers can work independently on separate branches of feature development, eventually merging them together.

Reduce Development cost

03

- Overall SCM is a huge aid to engineering teams that will lower development costs by allowing engineering resources to execute more efficiently.
- SCM is a must have in the modern age of software development. Professional teams use version control and your team should too.

CHITKARA UNIVERSITY

BEST PRACTICES

(SOURCE CODE MANAGEMENT)

01 Commit often

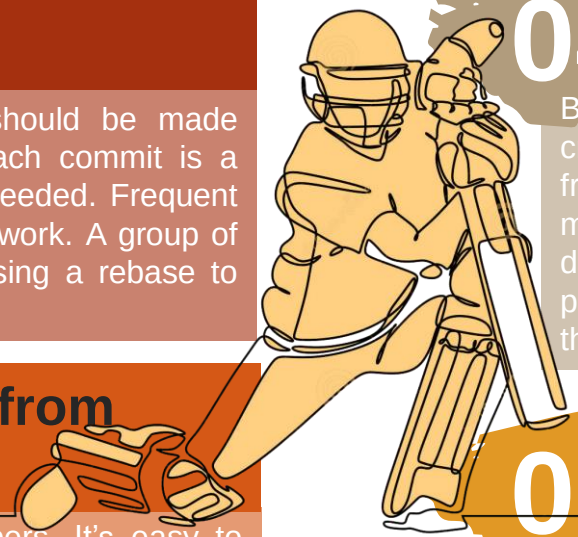
Commits are cheap and easy to make. They should be made frequently to capture updates to a code base. Each commit is a snapshot that the codebase can be reverted to if needed. Frequent commits give many opportunities to revert or undo work. A group of commits can be combined into a single commit using a rebase to clarify the development log.

02 Ensure you're working from latest version

SCM enables rapid updates from multiple developers. It's easy to have a local copy of the codebase fall behind the global copy. Make sure to git pull or fetch the latest code before making updates. This will help avoid conflicts at merge time.

03 Make detailed notes

Each commit has a corresponding log entry. At the time of commit creation, this log entry is populated with a message. It is important to leave descriptive explanatory commit log messages. These commit log messages should explain the "why" and "what" that encompass the commits content. These log messages become the canonical history of the project's development and leave a trail for future contributors to review.



04 Use Branches

Branching is a powerful SCM mechanism that allows developers to create a separate line of development. Branches should be used frequently as they are quick and inexpensive. Branches enable multiple developers to work in parallel on separate lines of development. These lines of development are generally different product features. When development is complete on a branch it is then merged into the main line of development.

05 Review changes before committing

SCM's offer a 'staging area'. The staging area can be used to collect a group of edits before writing them to a commit. The staging area can be used to manage and review changes before creating the commit snapshot. Utilizing the staging area in this manner provides a buffer area to help refine the contents of the commit.

06 Agree on a Workflow

By default SCMs offer very free form methods of contribution. It is important that teams establish shared patterns of collaboration. SCM workflows establish patterns and processes for merging branches. If a team doesn't agree on a shared workflow it can lead to inefficient communication overhead when it comes time to merge branches.

CHITKARA UNIVERSITY

SUMMARY

- SCM is an invaluable tool for modern software development.
- The best software teams use SCM and your team should too.
- SCM is very easy to set up on a new project and the return on investment is high.

CHITKARA UNIVERSITY

Version control system (VCS) for tracking changes in computer files

git

It is a Distributed Version Control tool that supports distributed non-linear workflows by providing data assurance for developing quality software.

INTRODUCTION | PERFORMANCE | SECURITY & FLEXIBILITY | VERSION CONTROL WITH GIT

CHITKARA UNIVERSITY

INTRODUCTION

git

is an integral part of any development project in the current IT world.

- By far, the most widely used modern version control system in the world today is Git.
- Git is a mature, actively maintained open source project originally developed in 2005 by Linus Torvalds, the famous creator of the Linux operating system kernel.
- A staggering number of software projects rely on Git for version control, including commercial projects as well as open source.
- Developers who have worked with Git are well represented in the pool of available software development talent and it works well on a wide range of operating systems and IDEs (Integrated Development Environments).
- Having a distributed architecture, Git is an example of a DVCS (hence Distributed Version Control System).
- Rather than have only one single place for the full version history of the software as is common in once-popular version control systems like CVS or Subversion (also known as SVN), in Git, every developer's working copy of the code is also a repository that can contain the full history of all changes.

CHITKARA UNIVERSITY

PERFORMANCE

- The raw performance characteristics of Git are very strong when compared to many alternatives.
- Committing new changes, branching, merging and comparing past versions are all optimized for performance.
- The algorithms implemented inside Git take advantage of deep knowledge about common attributes of real source code file trees, how they are usually modified over time and what the access patterns are.

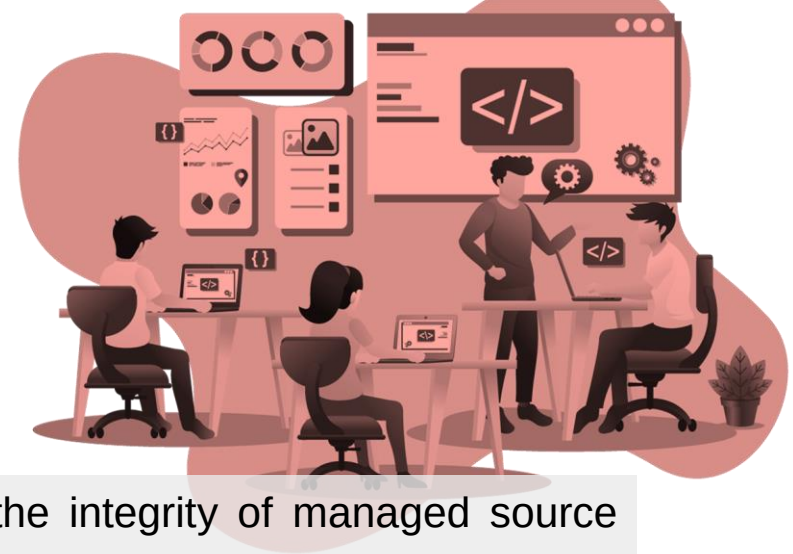
- Unlike some version control software, Git is not fooled by the names of the files when determining what the storage and version history of the file tree should be, instead, Git focuses on the file content itself.
- After all, source code files are frequently renamed, split, and rearranged.
- The object format of Git's repository files uses a combination of delta encoding (storing content differences), compression and explicitly stores directory contents and version metadata objects.

Being distributed enables significant performance benefits as well.

For example, say a developer, Alice, makes changes to source code, adding a feature for the upcoming 2.0 release, then commits those changes with descriptive messages. She then works on a second feature and commits those changes too. Naturally these are stored as separate pieces of work in the version history. Alice then switches to the version 1.3 branch of the same software to fix a bug that affects only that older version. The purpose of this is to enable Alice's team to ship a bug fix release, version 1.3.1, before version 2.0 is ready. Alice can then return to the 2.0 branch to continue working on new features for 2.0 and all of this can occur without any network access and is therefore fast and reliable. She could even do it on an airplane. When she is ready to send all of the individually committed changes to the remote repository, Alice can "push" them in one command.

CHITKARA UNIVERSITY

SECURITY

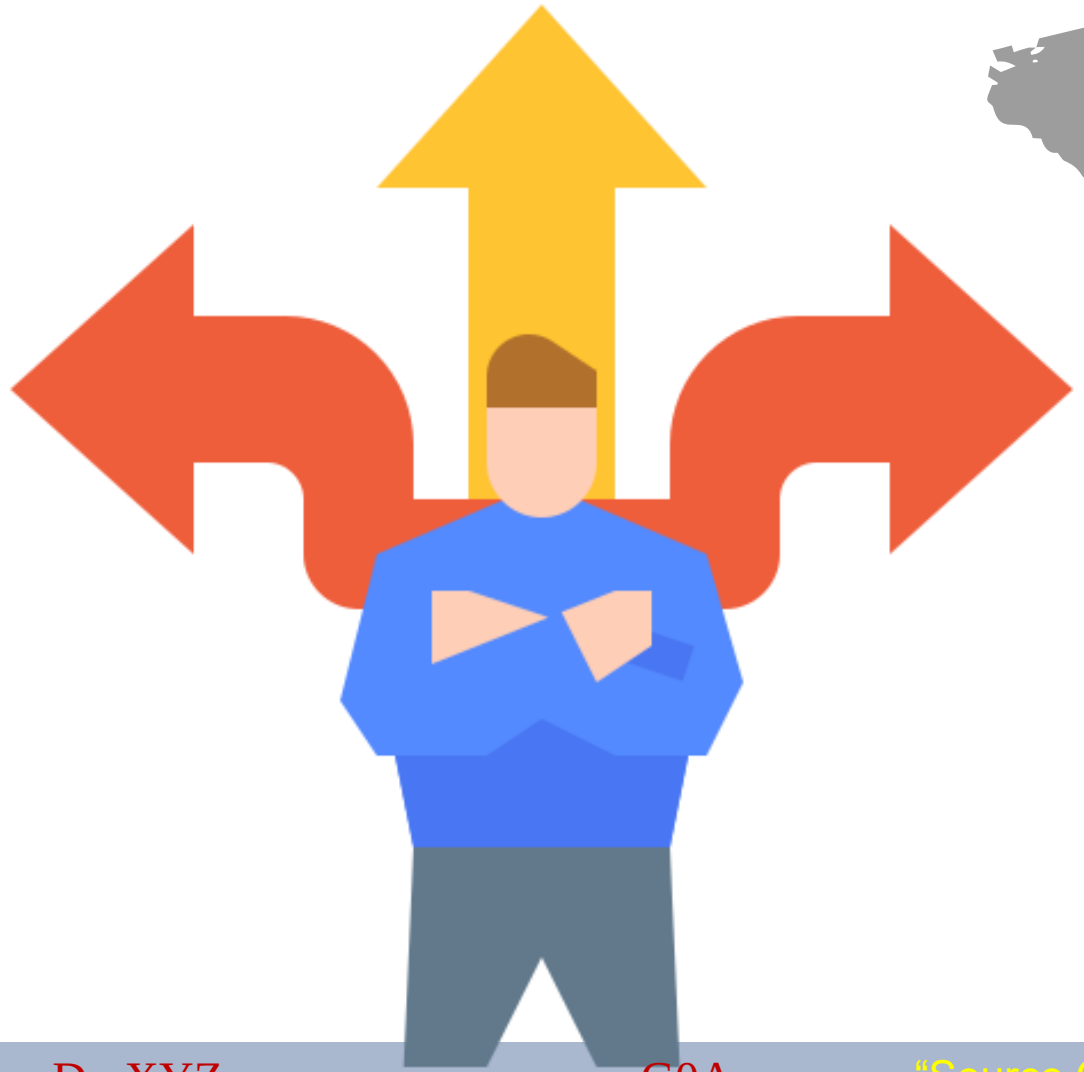


- Git has been designed with the integrity of managed source code as a top priority.
- The content of the files as well as the true relationships between files and directories, versions, tags and commits, all of these objects in the Git repository are secured with a cryptographically secure hashing algorithm called SHA1.
- This protects the code and the change history against both accidental and malicious change and ensures that the history is fully traceable.

- With Git, you can be sure you have an authentic content history of your source code.

Some other version control systems have no protections against secret alteration at a later date. This can be a serious information security vulnerability for any organization that relies on software development.

FLEXIBILITY



- One of Git's key design objectives is flexibility.
- Git is flexible in several respects: in support for various kinds of nonlinear development workflows, in its efficiency in both small and large projects and in its compatibility with many existing systems and protocols.

- Git has been designed to support branching and tagging as first-class citizens (unlike SVN) and operations that affect branches and tags (such as merging or reverting) are also stored as part of the change history.
- Not all version control systems feature this level of tracking.

VERSION CONTROL with GIT

- Git is the best choice for most software teams today. While every team is different and should do their own analysis, here are the main reasons why version control with Git is preferred over alternatives:

Git is good

Git has the functionality, performance, security and flexibility that most teams and individual developers need. These attributes of Git are detailed above. In side-by-side comparisons with most other alternatives, many teams find that Git is very favorable.

Git is a de facto standard

Git is the most broadly adopted tool of its kind. This makes Git attractive for the following reasons. At Atlassian, nearly all of our project source code is managed in Git.

Vast numbers of developers already have Git experience and a significant proportion of college graduates may have experience with only Git. While some organizations may need to climb the learning curve when migrating to Git from another version control system, many of their existing and future developers do not need to be trained on Git.

In addition to the benefits of a large talent pool, the predominance of Git also means that many third party software tools and services are already integrated with Git including IDEs, and our own tools like DVCS desktop client Sourcetree, issue and project tracking software, Jira, and code hosting service, Bitbucket.

If you are an inexperienced developer wanting to build up valuable skills in software development tools, when it comes to version control, Git should be on your list.

VERSION CONTROL with GIT

- Git is the best choice for most software teams today. While every team is different and should do their own analysis, here are the main reasons why version control with Git is preferred over alternatives:

Git is a quality open source project

Git is a very well supported open source project with over a decade of solid stewardship. The project maintainers have shown balanced judgment and a mature approach to meeting the long term needs of its users with regular releases that improve usability and functionality. The quality of the open source software is easily scrutinized and countless businesses rely heavily on that quality.

Git enjoys great community support and a vast user base. Documentation is excellent and plentiful, including books, tutorials and dedicated web sites. There are also podcasts and video tutorials.

Being open source lowers the cost for hobbyist developers as they can use Git without paying a fee. For use in open-source projects, Git is undoubtedly the successor to the previous generations of successful open source version control systems, SVN and CVS.

VERSION CONTROL with GIT

- One common criticism of Git is that it can be difficult to learn.
- Some of the terminology in Git will be novel to newcomers and for users of other systems, the Git terminology may be different, for example, revert in Git has a different meaning than in SVN or CVS.
- Nevertheless, Git is very capable and provides a lot of power to its users.
- Learning to use that power can take some time, however once it has been learned, that power can be used by the team to increase their development speed.

- For those teams coming from a non-distributed VCS, having a central repository may seem like a good thing that they don't want to lose.
- However, while Git has been designed as a distributed version control system (DVCS), with Git, you can still have an official, canonical repository where all changes to the software must be stored.
- With Git, because each developer's repository is complete, their work doesn't need to be constrained by the availability and performance of the "central" server. During outages or while offline, developers can still consult the full project history.
- Because Git is flexible as well as being distributed, you can work the way you are accustomed to but gain the additional benefits of Git, some of which you may not even realise you're missing.

CHITKARA UNIVERSITY

git

FOR EVERYONE

Switching from a centralized version control system to Git changes the way your development team creates software. And, if you're a company that relies on its software for mission-critical applications, altering your development workflow impacts your entire business.

Here, we'll discuss how Git benefits each aspect of your organization, from your development team to your marketing team, and everything in between. By the end of this article, it should be clear that Git isn't just for agile software development—it's for agile business.

01 Developers

02 Marketing

03 Product Management

04 Designers

05 Customer Support

06 Human Resources

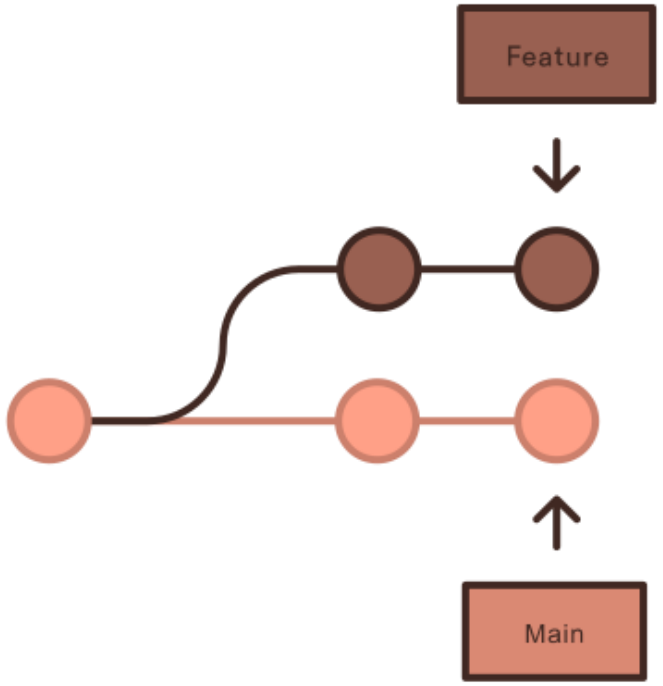
07 Managing a budget

❖Feature Branch Workflow

- One of the biggest advantages of Git is its branching capabilities.
- Unlike centralized version control systems, Git branches are cheap and easy to merge.
- This facilitates the feature branch workflow popular with many Git users.

- Feature branches provide an isolated environment for every change to your codebase.
- When a developer wants to start working on something—no matter how big or small—they create a new branch.
- This ensures that the main branch always contains production-quality code.

- Using feature branches is not only more reliable than directly editing production code, but it also provides organizational benefits.
- They let you represent development work at the same granularity as the your agile backlog.
  - For example, you might implement a policy where each Jira ticket is addressed in its own feature branch.



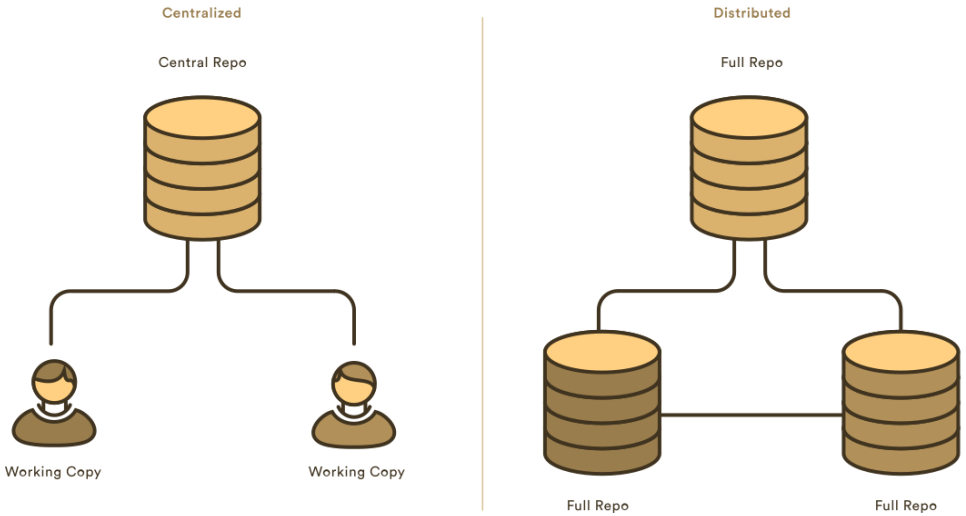
❖Distributed Development

- In SVN, each developer gets a working copy that points back to a single central repository.
- Git, however, is a distributed version control system. Instead of a working copy, each developer gets their own local repository, complete with a full history of commits.

- Having a full local history makes Git fast, since it means you don't need a network connection to create commits, inspect previous versions of a file, or perform diffs between commits.

- Distributed development also makes it easier to scale your engineering team.
- If someone breaks the production branch in SVN, other developers can't check in their changes until it's fixed. With Git, this kind of blocking doesn't exist.
- Everybody can continue going about their business in their own local repositories.

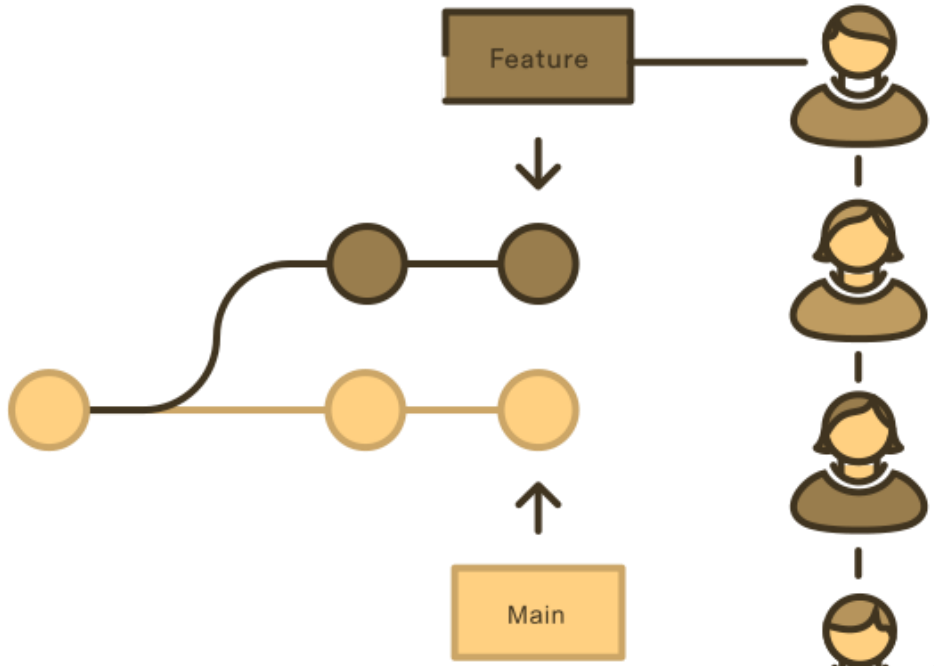
- And, similar to feature branches, distributed development creates a more reliable environment.
- Even if a developer obliterates their own repository, they can simply clone someone else's and start anew.



❖Pull Requests

- Many source code management tools such as Bitbucket enhance core Git functionality with pull requests.
- A pull request is a way to ask another developer to merge one of your branches into their repository.
- This not only makes it easier for project leads to keep track of changes, but also lets developers initiate discussions around their work before integrating it with the rest of the codebase.

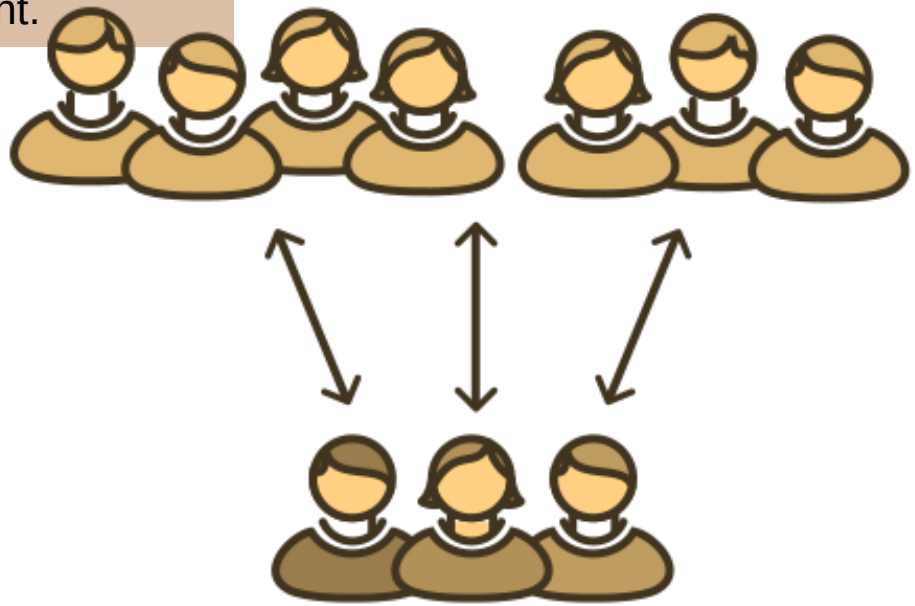
- Since they're essentially a comment thread attached to a feature branch, pull requests are extremely versatile.
- When a developer gets stuck with a hard problem, they can open a pull request to ask for help from the rest of the team.
- Alternatively, junior developers can be confident that they aren't destroying the entire project by treating pull requests as a formal code review.



CHITKARA UNIVERSITY git for DEVELOPERS

❖Community

- In many circles, Git has come to be the expected version control system for new projects.
- If your team is using Git, odds are you won't have to train new hires on your workflow, because they'll already be familiar with distributed development.



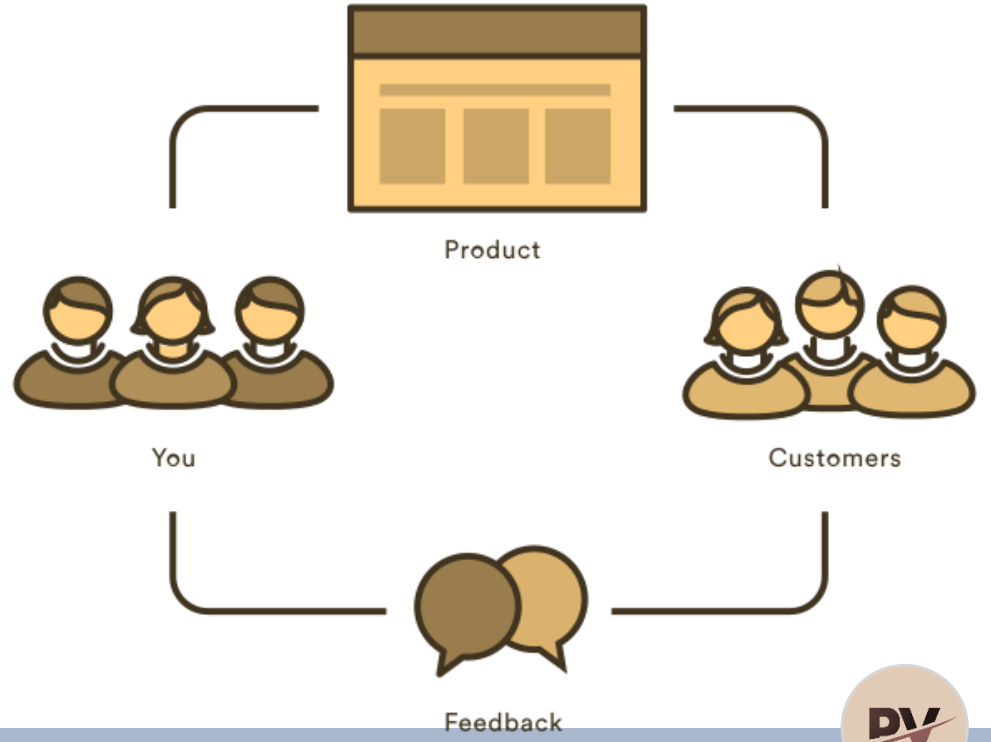
- In addition, Git is very popular among open source projects.
- This means it's easy to leverage 3rd-party libraries and encourage others to fork your own open source code.

❖Faster Release Cycle

- The ultimate result of feature branches, distributed development, pull requests, and a stable community is a faster release cycle.
- These capabilities facilitate an agile workflow where developers are encouraged to share smaller changes more frequently.
- In turn, changes can get pushed down the deployment pipeline faster than the monolithic releases common with centralized version control systems.

- As you might expect, Git works very well with continuous integration and continuous delivery environments.
- Git hooks allow you to run scripts when certain events occur inside of a repository, which lets you automate deployment to your heart's content.
- You can even build or deploy code from specific branches to different servers.

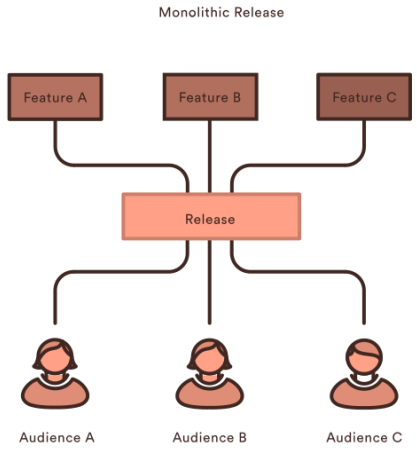
- For example, you might want to configure Git to deploy the most recent commit from the develop branch to a test server whenever anyone merges a pull request into it. Combining this kind of build automation with peer review means you have the highest possible confidence in your code as it moves from development to staging to production.

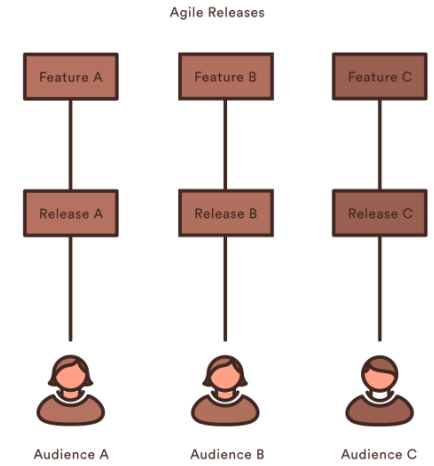


CHITKARA UNIVERSITY git for MARKETING

- To understand how switching to Git affects your company's marketing activities, imagine your development team has three distinct changes scheduled for completion in the next few weeks:
  - The entire team is finishing up a game-changing feature that they've been working on for the last 6 months.
  - Mary is implementing a smaller, unrelated feature that only impacts existing customers.
  - Rick is making some much-needed updates to the user interface.

- If you're using a traditional development workflow that relies on a centralized VCS, all of these changes would probably be rolled up into a single release.
- Marketing can only make one announcement that focuses primarily on the game-changing feature, and the marketing potential of the other two updates is effectively ignored.





- The shorter development cycle facilitated by Git makes it much easier to divide these into individual releases.
- This gives marketers more to talk about, more often.
- In the above scenario, marketing can build out three campaigns that revolve around each feature, and thus target very specific market segments.

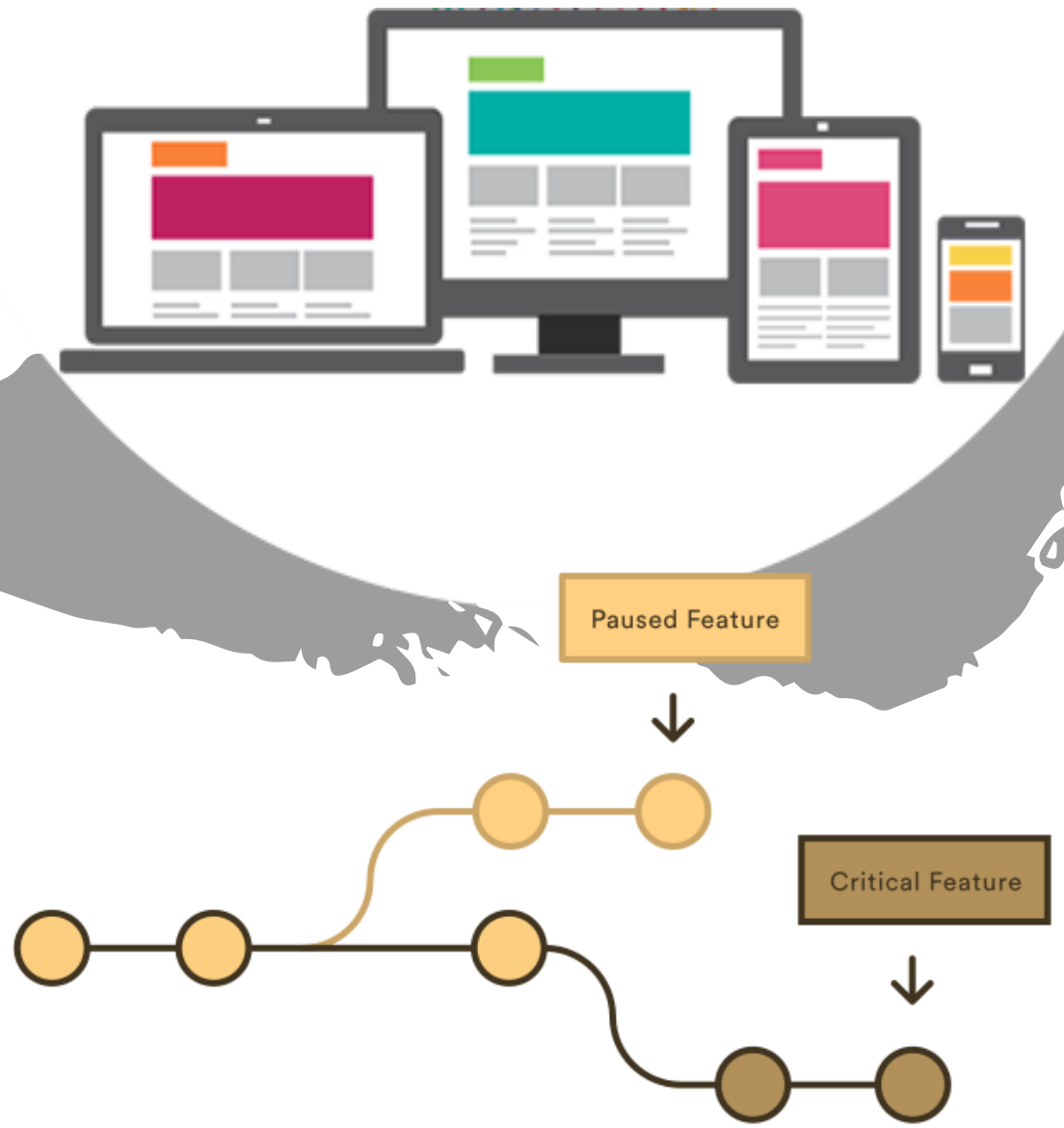
- For instance, they might prepare a big PR push for the game changing feature, a corporate blog post and newsletter blurb for Mary's feature, and some guest posts about Rick's underlying UX theory for sending to external design blogs.
- All of these activities can be synchronized with a separate release.

CHITKARA UNIVERSITY git for PRODUCT MANAGEMENT

- The benefits of Git for product management is much the same as for marketing.
- More frequent releases means more frequent customer feedback and faster updates in reaction to that feedback.
- Instead of waiting for the next release 8 weeks from now, you can push a solution out to customers as quickly as your developers can write the code.

- The feature branch workflow also provides flexibility when priorities change.
- For instance, if you're halfway through a release cycle and you want to postpone one feature in lieu of another time-critical one, it's no problem.
- That initial feature can sit around in its own branch until engineering has time to come back to it.

- This same functionality makes it easy to manage innovation projects, beta tests, and rapid prototypes as independent codebases.



CHITKARA UNIVERSITY

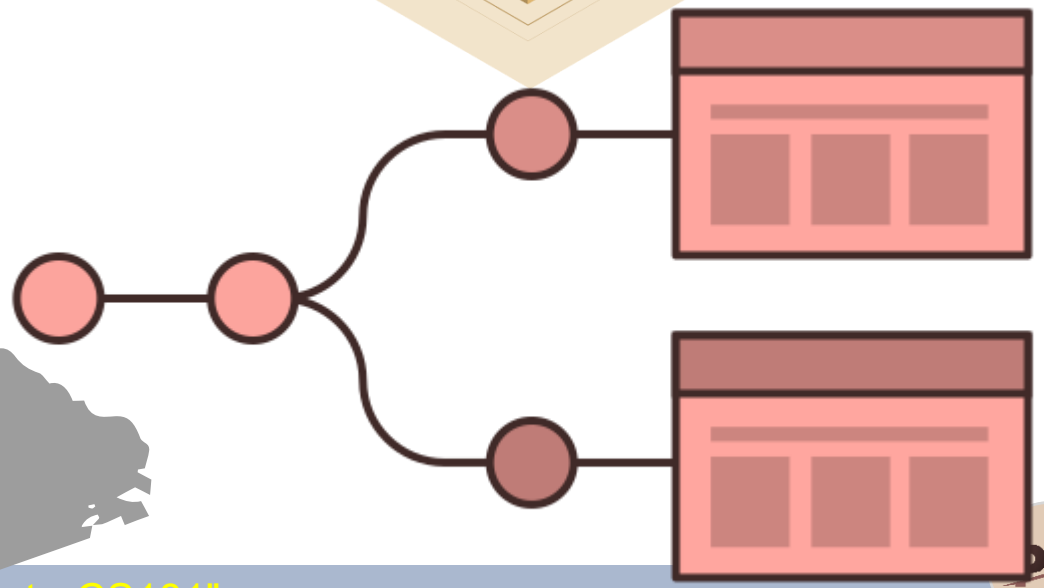
git for DESIGNERS

- Feature branches lend themselves to rapid prototyping. Whether your UX/UI designers want to implement an entirely new user flow or simply replace some icons, checking out a new branch gives them a sandboxed environment to play with.
- This lets designers see how their changes will look in a real working copy of the product without the threat of breaking existing functionality.

- Encapsulating user interface changes like this makes it easy to present updates to other stakeholders.
- For example, if the director of engineering wants to see what the design team has been working on, all they have to do is tell the director to check out the corresponding branch.

- Pull requests take this one step further and provide a formal place for interested parties to discuss the new interface.
- Designers can make any necessary changes, and the resulting commits will show up in the pull request. This invites everybody to participate in the iteration process.

- Perhaps the best part of prototyping with branches is that it's just as easy to merge the changes into production as it is to throw them away.
- There's no pressure to do either one. This encourages designers and UI developers to experiment while ensuring that only the best ideas make it through to the customer.



CHITKARA UNIVERSITY git for CUSTOMER SUPPORT

- Customer support and customer success often have a different take on updates than product managers.
- When a customer calls them up, they're usually experiencing some kind of problem.
- If that problem is caused by your company's software, a bug fix needs to be pushed out as soon as possible.

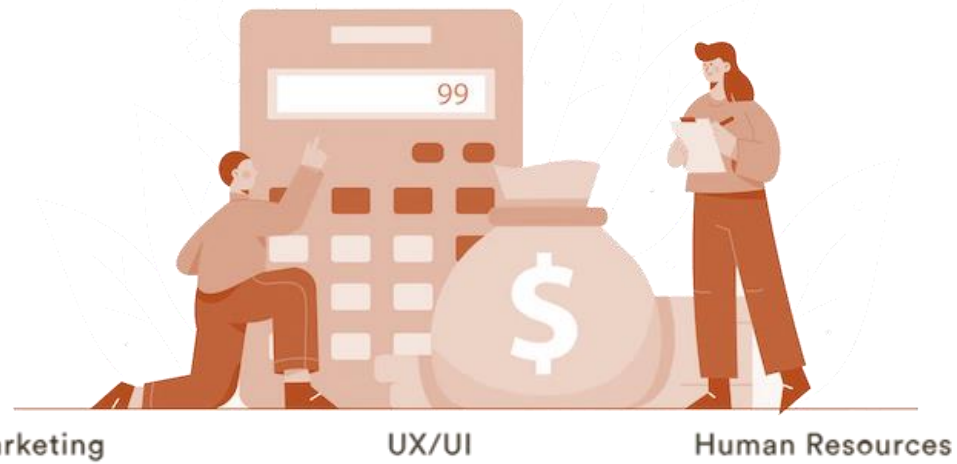
- Git's streamlined development cycle avoids postponing bug fixes until the next monolithic release.
- A developer can patch the problem and push it directly to production.
- Faster fixes means happier customers and fewer repeat support tickets.
- Instead of being stuck with, "Sorry, we'll get right on that" your customer support team can start responding with "We've already fixed it!

git for HUMAN RESOURCES

- To a certain extent, your software development workflow determines who you hire.
- It always helps to hire engineers that are familiar with your technologies and workflows, but using Git also provides other advantages.

- Employees are drawn to companies that provide career growth opportunities, and understanding how to leverage Git in both large and small organizations is a boon to any programmer.
- By choosing Git as your version control system, you're making the decision to attract forward-looking developers.

CHITKARA UNIVERSITY git for ANYONE MANAGING A BUDGET



- Git is all about efficiency.
- For developers, it eliminates everything from the time wasted passing commits over a network connection to the man hours required to integrate changes in a centralized version control system.
- It even makes better use of junior developers by giving them a safe environment to work in.
- All of this affects the bottom line of your engineering department.

- But, don't forget that these efficiencies also extend outside your development team.
- They prevent marketing from pouring energy into collateral for features that aren't popular.
- They let designers test new interfaces on the actual product with little overhead.
- They let you react to customer complaints immediately.

- Being agile is all about finding out what works as quickly as possible, magnifying efforts that are successful, and eliminating ones that aren't.
- Git serves as a multiplier for all your business activities by making sure every department is doing their job more efficiently.

CHITKARA UNIVERSITY

STEP OF INSTALLATION

INSTALLATION Steps for

(Git for Windows stand-alone installer)

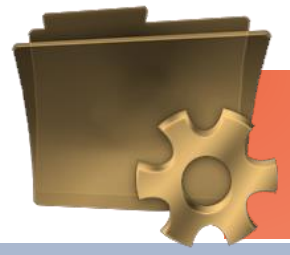
1. Download the latest Git for Windows installer. https://gitforwindows.org

2. When you've successfully started the installer, you should see the Git Setup wizard screen. Follow the Next and Finish prompts to complete the installation. The default options are pretty sensible for most users.

3. Open a Command Prompt (or Git Bash if during installation you elected not to use Git from the Windows Command Prompt).

4. Run the following commands to configure your Git username and email using the following commands, replacing Parag name with your own.
These details will be associated with any commits that you create:

```
$ git config --global user.name “Parag India Paris" $ git config --global user.email “parag.verma@chitkara.edu.in"
```



Optional: Install the Git credential helper on Windows
Bitbucket supports pushing and pulling over HTTP to your remote Git repositories on Bitbucket. Every time you interact with the remote repository, you must supply a username/password combination. You can store these credentials, instead of supplying the combination every time, with the Git Credential Manager for Windows.

INSTALLATION Steps for Linux

Debian / Ubuntu (apt-get)

From your shell, install Git using apt-get:

```
$ sudo apt-get update
$ sudo apt-get install git
```

Verify the installation was successful by typing git --version:

```
$ git --version
```

Configure your Git username and email using the following commands, replacing Emma's name with your own.
These details will be associated with any commits that you create:

```
$ git config --global user.name “Parag India"
$ git config --global user.email “parag.verma@chitkara.edu.in"
```

CHITKARA UNIVERSITY

INSTALLATION Steps for Mac OS

There are several ways to install Git on a Mac. In fact, if you've installed XCode (or it's Command Line Tools), Git may already be installed. To find out, open a terminal and enter **“git –version”.**

Apple actually maintain and ship their own fork of Git, but it tends to lag behind mainstream Git by several major versions. You may want to install a newer version of Git using one of the methods below:

Git for Mac Installer

(The easiest way to install Git on a Mac is via the stand-alone installer:)

Download the latest Git for Mac installer.

- https://sourceforge.net/projects/git-osx-installer/files/

Follow the prompts to install Git.

Open a terminal and verify the installation was successful by typing **“git --version”**

Configure your Git username and email using the following commands, replacing Parag name with your own. These details will be associated with any commits that you create:

```
$ git config --global user.name “Parag India"
$ git config --global user.email “parag.verma@chitkara.edu.in"
```

(Optional) To make Git remember your username and password when working with HTTPS repositories, configure the git-credential-osxkeychain helper.

CHITKARA UNIVERSITY

INSTALLATION Steps for Mac OS

There are several ways to install Git on a Mac. In fact, if you've installed XCode (or it's Command Line Tools), Git may already be installed. To find out, open a terminal and enter **“git –version”.**

Apple actually maintain and ship their own fork of Git, but it tends to lag behind mainstream Git by several major versions. You may want to install a newer version of Git using one of the methods below:

Install Git with Homebrew

(If you have installed Homebrew to manage packages on OS X, you can follow these instructions to install Git:)

1

Open your terminal and install Git using Homebrew:

$ brew install git

2

Verify the installation was successful by typing which **“git --version”**

3

- Configure your Git username and email using the following commands, replacing Parag name with your own.
- These details will be associated with any commits that you create:

```
$ git config --global user.name “Parag India"
$ git config --global user.email “parag.verma@chitkara.edu.in"
```

(Optional) To make Git remember your username and password when working with HTTPS repositories, install the git-credential-osxkeychain helper.

CHITKARA UNIVERSITY

SECURE SHELL

git

An SSH key is an access credential for the SSH (secure shell) network protocol.

What is SSH

SSH uses a pair of keys to initiate a secure handshake between remote parties.

Create SSH

Generated through a public key cryptographic algorithm, the most common being RSA or DSA.

Generate SSH for OSs

Both OsX and Linux operating systems have comprehensive modern terminal applications that ship with the SSH suite installed.

INTRODUCTION to SSH KEY

- An SSH key is an access credential for the SSH (secure shell) network protocol.
- This authenticated and encrypted secure network protocol is used for remote communication between machines on an unsecured open network.
- SSH is used for remote file transfer, network management, and remote operating system access.
- The SSH acronym is also used to describe a set of tools used to interact with the SSH protocol.

- SSH uses a pair of keys to initiate a secure handshake between remote parties.
- The key pair contains a public and private key.
- The private vs public nomenclature can be confusing as they are both called keys.
- It is more helpful to think of the public key as a "lock" and the private key as the "key".
- You give the public 'lock' to remote parties to encrypt or 'lock' data. This data is then opened with the 'private' key which you hold in a secure place.

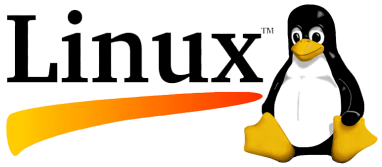
How To Create

- SSH keys are generated through a public key cryptographic algorithm, the most common being RSA or DSA.
- At a very high level SSH keys are generated through a mathematical formula that takes 2 prime numbers and a random seed variable to output the public and private key.
- This is a one-way formula that ensures the public key can be derived from the private key but the private key cannot be derived from the public key.

- SSH keys are created using a key generation tool.
- The SSH command line tool suite includes a keygen tool.
- Most git hosting providers offer guides on how to create an SSH Key.

CHITKARA UNIVERSITY

GENERATE An git



Mac OS

- Both OsX and Linux operating systems have comprehensive modern terminal applications that ship with the SSH suite installed.
- The process for creating an SSH key is the same between them.

1 Execute the following to begin the key creation

ssh-keygen -t rsa -b 4096 -C "your_email@example.com"

This command will create a new SSH key using the email as a label

2 You will then be prompted to "Enter a file in which to save the key."

Enter a file in which to save the key (/Users/you/.ssh/id_rsa): [Press enter]

You can specify a file location or press “Enter” to accept the default file location.

3 The next prompt will ask for a secure passphrase.

A passphrase will add an additional layer of security to the SSH and will be required anytime the SSH key is used. If someone gains access to the computer that private keys are stored on, they could also gain access to any system that uses that key. Adding a passphrase to keys will prevent this scenario.

Enter passphrase (empty for no passphrase): [Type a passphrase]
Enter same passphrase again: [Type passphrase again]

At this point, a new SSH key will have been generated at the previously specified file path.

4 Add the new SSH key to the ssh-agent

The ssh-agent is another program that is part of the SSH toolsuite. The ssh-agent is responsible for holding private keys. Think of it like a keychain. In addition to holding private keys it also brokers requests to sign SSH requests with the private keys so that private keys are never passed around unsecurly.

Before adding the new SSH key to the ssh-agent first ensure the ssh-agent is running by executing:

$ eval "$(ssh-agent -s)"
> Agent pid 59566

Once the ssh-agent is running the following command will add the new SSH key to the local SSH agent.

ssh-add -K /Users/you/.ssh/id_rsa

The new SSH key is now registered and ready to use!

GENERATE An

- Windows environments do not have a standard default unix shell.
- External shell programs will need to be installed for to have a complete keygen experience.
- The most straight forward option is to utilize Git Bash.
- Once Git Bash is installed the same steps for Linux and Mac can be followed within the Git Bash shell.

Windows Linux Subsystem

- Modern windows environments offer a windows linux subsystem.
- The windows linux subsystem offers a full linux shell within a traditional windows environment.
- If a linux subsystem is available the same steps previously discussed for Linux and Mac can be followed with in the windows linux subsystem.

CHITKARA UNIVERSITY

git

ARCHIVE

- An archive file combines multiple files into a single file.
- An archive file can then be extracted to reproduce the individual files.
- Git is incredibly powerful at preserving history and team collaboration; however, archive files remove the overhead of Git's metadata and can be simpler to distribute to other users or preserve in long term cold storage.

An archive file combines multiple files into a single file.

Export a git project

SSH uses a pair of keys to initiate a secure handshake between remote parties.

Ability of git archive

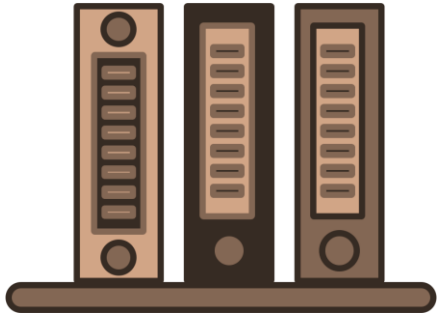
Generated through a public key cryptographic algorithm, the most common being RSA or DSA.

Example

Both OsX and Linux operating systems have comprehensive modern terminal applications that ship with the SSH suite installed.

CHITKARA UNIVERSITY

git ARCHIVE



What does git archive do?

The git archive command is a Git command line utility that will create an archive file from specified Git Refs like, commits, branches, or trees. git archive accepts additional arguments that will alter the archive output.

Git export examples

A most basic ~git archive~ example follows

git archive --format=tar HEAD

- This command when executed will create an archive from the current HEAD ref of the repository.
- By default, git archive will stream the archive output to the ephemeral stdout stream.
- You will need to capture this output stream to a permanent file.
- You can specify a permanent file by using git archives output option or using the operating systems stdout redirection.

git archive --output=./example_repo_archive.tar --format=tar HEAD

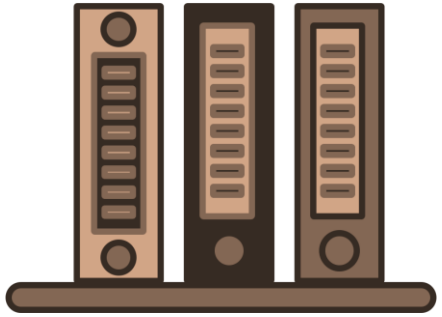
- The proceeding example will create a new archive and store it in the exmaple_repo_archive.tar file.
- The previous examples have both created uncompressed archive output.
- This is denoted by the --format=tar option.
- The format option also accepts popular compressed file formats zip and tar.gz.
- Passing one of these format options will produce a compressed archive.
- If a format value is not passed it will be inferred from any --output option passed.

git archive --output=./example_repo_archive.tar.gz --format=tar HEAD ./build

- A partial archives of the repository can be created by passing a path argument.
- This example adds a ./build path argument to the archive command.
- This command will output an archive containing only files stored under the ./build directory

CHITKARA UNIVERSITY

git ARCHIVE



Options

- The previous examples demonstrated some of the most frequently used git-archive use cases.
- The following are extended options that can be passed to git-archive.

--prefix=<prefix>/

- The prefix options prepends a path to each file in an archive.
- This can be helpful to ensure the archive contents get extracted in a unique namespace.

--remote=<repo>

- The remote option expects a remote repository URL.
- When invoked with the remote option, git-archive will fetch the remote repository and create an archive from the specified ref if it's available on the remote.

Configuration

- There are a few global Git configuration values that ~git archive~ will respect. These values can be set using the [git config][link to git config] utility.

tar.umask

- The unmask configuration option is used to specify unix level permission bit restriction on the output archive file.

tar.<format>.command

- This configuration option allows specification of a custom shell command that the git-archive output will be run through.
- This is similar to omitting the --output option and piping the stdout stream from ~git archive~ to a custom tool.
- This enables fixed custom archive post-processing.

tar.<format>.remote

- If enabled this allows remote clients to fetch archives of type format.

CHITKARA UNIVERSITY

gitOps

GitOps is an operational framework that takes DevOps best practices used for application development such as version control, collaboration, compliance, and CI/CD, and applies them to infrastructure automation.

INTRODUCTION & HISTORY	BENIFITS	WORKING	EXAMPLES

CHITKARA UNIVERSITY

gitOps

- Many organizations now see DevOps as part of their digital transformation strategy, since it encourages a culture of shared responsibility, transparency, and faster feedback.
- Yet as the gap between development and operations teams shrinks, so do the processes.

- So it goes with Git, the most widely used version control system in the world today.
- As companies embrace DevOps methodologies, so to the tools, which has created an evolution to GitOps, a set of practices that allow developers to perform more IT operations-related tasks.

Is GitOps the NEXT BIG THING In DevOps

?

CHITKARA UNIVERSITY

WHAT is GitOps

- GitOps is an operational framework that takes DevOps best practices used for application development such as version control, collaboration, compliance, and CI/CD tooling, and applies them to infrastructure automation.
- While the software development lifecycle has been automated, infrastructure has remained a largely manual process that requires specialized teams.
- With the demands made on today's infrastructure, it has become increasingly crucial to implement infrastructure automation.
- Modern infrastructure needs to be elastic so that it can effectively manage cloud resources that are needed for continuous deployments.

- Modern applications are developed with speed and scale in mind.
- Organizations with a mature DevOps culture can deploy code to production hundreds of times per day.
- DevOps teams can accomplish this through development best practices such as version control, code review, and CI/CD pipelines that automate testing and deployments.

- GitOps is used to automate the process of provisioning infrastructure.
- Similar to how teams use application source code, operations teams that adopt GitOps use configuration files stored as code (infrastructure as code).
- GitOps configuration files generate the same infrastructure environment every time it's deployed, just as application source code generates the same application binaries every time it's built.

ABOUT gitOps

CHITKARA UNIVERSITY

GitOps into PRACTICE

GitOps = IaC + MRs + CI/CD

GitOps is not a single product, plugin, or platform. GitOps workflows help teams manage IT infrastructure through processes they already use in application development.

IaC (Infrastructure as code)

- GitOps uses a Git repository as the single source of truth for infrastructure definitions.
- Git is an open source version control system that tracks code management changes, and a Git repository is a .git folder in a project that tracks all changes made to files in a project over time.
- Infrastructure as code (IaC) is the practice of keeping all infrastructure configuration stored as code.
- The actual desired state may or may not be not stored as code (e.g., number of replicas or pods).

MRs (Merge Requests)

- GitOps uses merge requests (MRs) as the change mechanism for all infrastructure updates.
- The MR is where teams can collaborate via reviews and comments and where formal approvals take place.
- A merge commits to your main (or trunk) branch and serves as an audit log.

CI/CD (Continuous integration and Continuous delivery)

- GitOps automates infrastructure updates using a Git workflow with continuous integration (CI) and continuous delivery (CI/CD).
- When new code is merged, the CI/CD pipeline enacts the change in the environment.
- Any configuration drift, such as manual changes or errors, is overwritten by GitOps automation so the environment converges on the desired state defined in Git.
- GitLab uses CI/CD pipelines to manage and implement GitOps automation, but other forms of automation, such as definitions operators, can be used as well.

CHITKARA UNIVERSITY

BENEFITS of gitOps

- GitOps shares many of the same benefits as an agile feature branch software development workflow.
- The first major benefit is ease of adoption due to the usage of common tools. Git is the de facto standard of version control systems and is a common software development tool for most developers and software teams.
- This makes it easy for developers familiar with Git to become cross functional contributors and participate in GitOps.

- Using a version control system lets a team track all modifications to the configuration of a system.
- This gives a “source of truth” and valuable audit trail to review if something breaks or behaves unexpectedly.
- Teams can review the GitOps history and see when a regression was introduced. Additionally this audit trail can be used as a reference for compliance or security auditing.
- The GitOps history can be used as proof when things like encryption certificates are modified or updated.

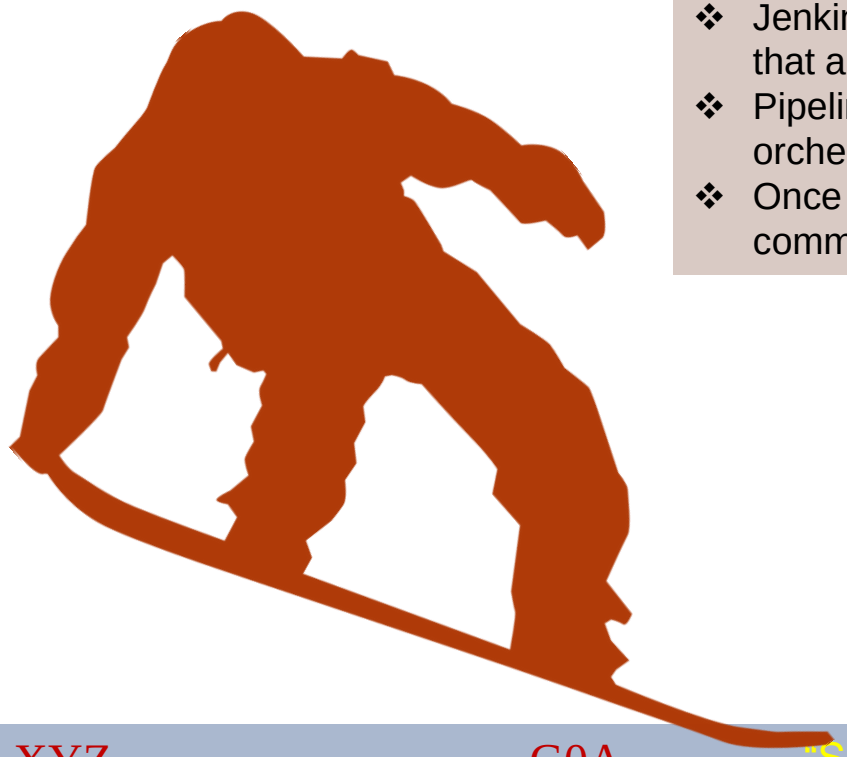
- GitOps brings transparency and clarity to an organization's infrastructure needs around a central repo.
- Containing all systems configurations in a central repository helps scale contribution input from team members.
- Pull requests made through hosted Git services like Bitbucket have rich tools for code review and discussion commentary.
- This builds passive communication loops that allows the full engineering team to observe and monitor infrastructure changes.

- GitOps can greatly increase productivity for a DevOps team.
- It allows them to quickly experiment with new infrastructure configurations.
- If the new changes don’t behave as expected, a team can use Git history to revert changes to a known good state.
- This is incredibly powerful since it enables the familiar “undo” functionality in a complicated infrastructure.

CHITKARA UNIVERSITY

WORKING of gitops

- GitOps procedures are performed by an underlying orchestration system.
- GitOps itself is an agnostic best practice pattern.
- Many popular GitOps solutions today primarily use Kubernetes as the orchestration system.
- Some alternative GitOps tool sets are coming to market that support direct Terraform manipulation.



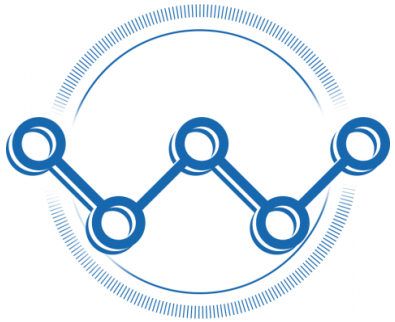
- To achieve a full GitOps install, a pipeline platform is required.
- Jenkins, Bitbucket Pipelines, or CircleCi are some popular pipeline tools that are complementary to GitOps.
- Pipelines automate and bridge the gap between Git pull requests and the orchestration system.
- Once pipeline hooks are established and triggered from pull requests, commands are executed to the orchestration piece.

- A new pattern or component that is specifically introduced with GitOps is the GitOps “operator,” which is a mechanism that sits between the pipeline and the orchestration system.
- A pull request starts the pipeline that then triggers the operator.
- The operator examines the state of the repository and the start of the orchestration and syncs them.
- The operator is the magic component of GitOps.

CHITKARA UNIVERSITY

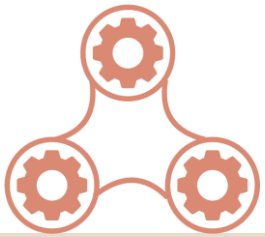
EXAMPLEs of gitOps

Imagine a team identified a performance bottleneck or a spike in traffic and the team notices the load balancer is not working as expected. They look into the GitOps repo that holds the infrastructure configuration and find a specific file that configures and deploys the load balancer. They can review it on their online Git hosting site. After some review and discussion they identify that some of the configuration values for the load balancer are not optimal and need to be adjusted.



A member of the team opens up a new pull request that optimizes the load balancer values. The pull request is reviewed and approved by a second team member and merged into the repository. The merge kicks off a GitOps pipeline, which triggers the GitOps operator. The operator sees the load balancer configuration was changed. It confirms with the systems orchestration tool that this does not match what is live on the teams cluster.

The operator signals the orchestration system to update the load balancer configuration. The orchestrator handles the rest and automatically deploys the newly configured load balancer. The team then monitors the newly updated live system to see it return to a healthy state. This is an ideal GitOps scenario. Let's expand on it further to demonstrate GitOps utility.

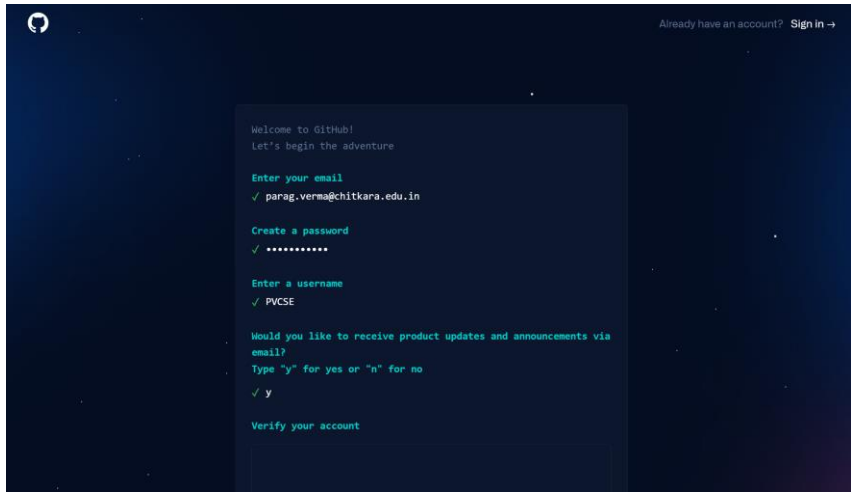


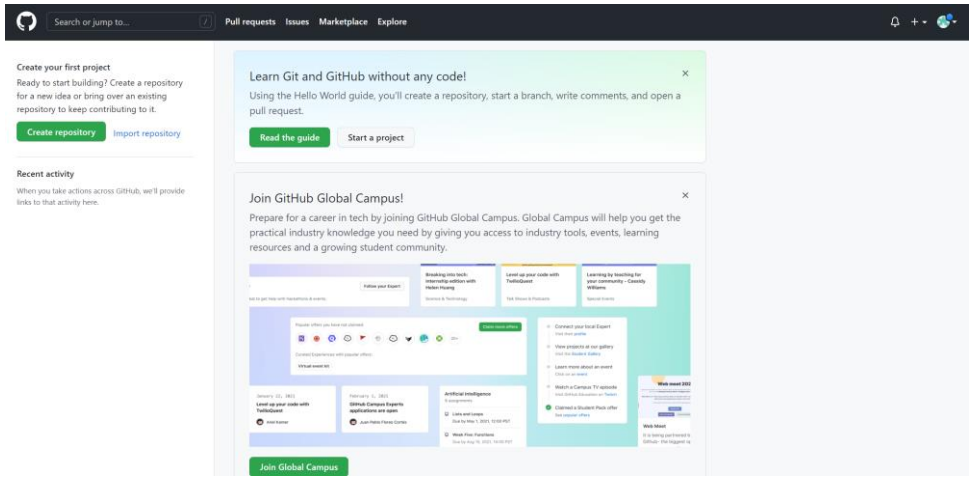
Let's imagine that instead of slightly tweaking the load balancer values to be more optimal, the team makes an aggressive decision to deploy an entirely new load balancer type. They feel the current load balancer is fundamentally flawed and want to try an alternative option. The workflow is the same as the value tweak. The team creates a pull request that introduces an entirely new load balancer configuration and deletes the old configuration. It is approved and deployed through the pipeline.

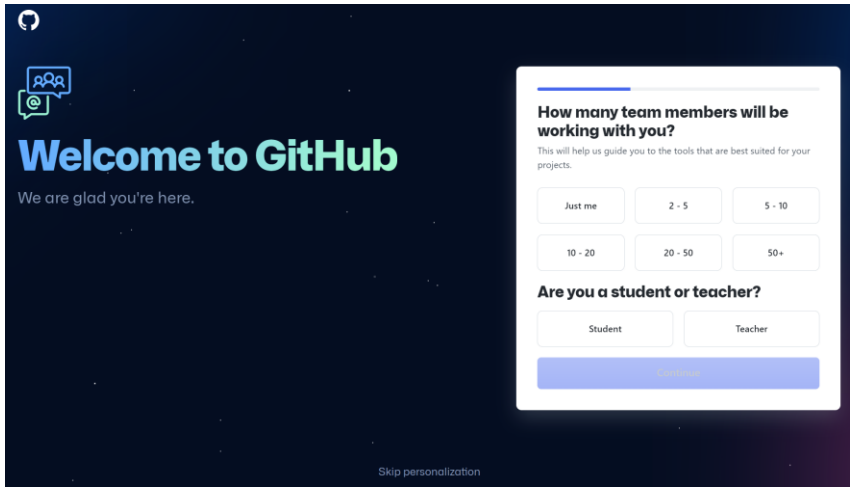
Unfortunately the team finds that this new type of load balancer is incompatible with some other services within their cluster. The new load balancer causes critical traffic failures and halts user operations. Luckily because the team has a complete GitOps pipeline they can quickly undo these load balancer changes. The team will make another pull request that reverts the repository back to the old known functional load balancer. This again will be noted by the GitOps pipeline and automatically deployed. It will rapidly improve the infrastructure and improves the reliability score of the team.

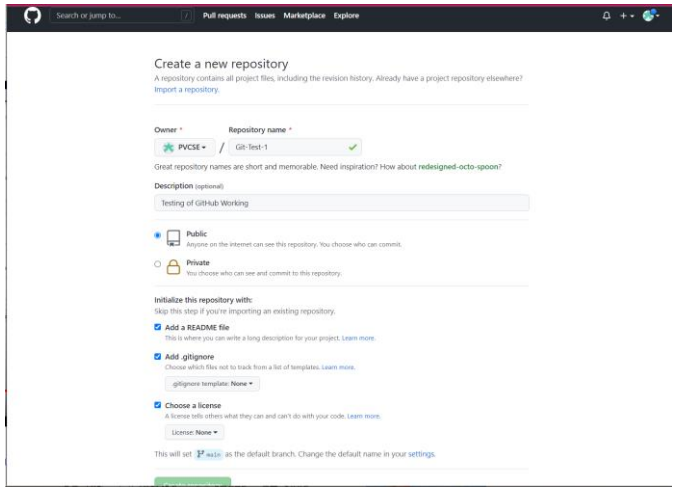
CHITKARA UNIVERSITY

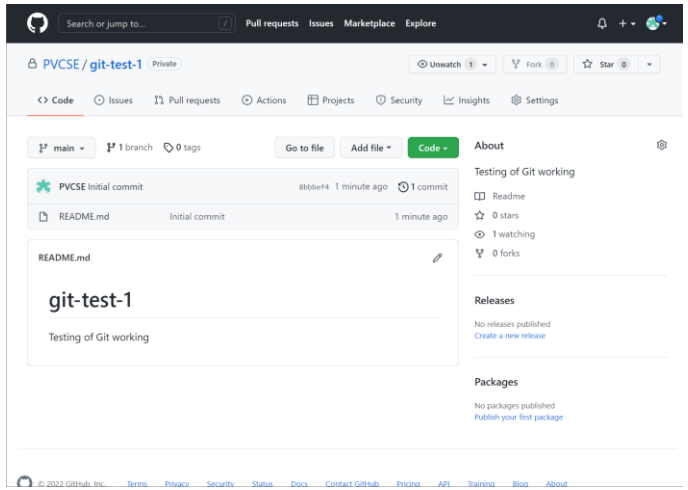
SignUp











CHITKARA UNIVERSITY

git

REPOSITORY

A Git repository is a virtual storage of your project. It allows you to save versions of your code, which you can access when needed.

SETTING UP

SAVING CHANGES

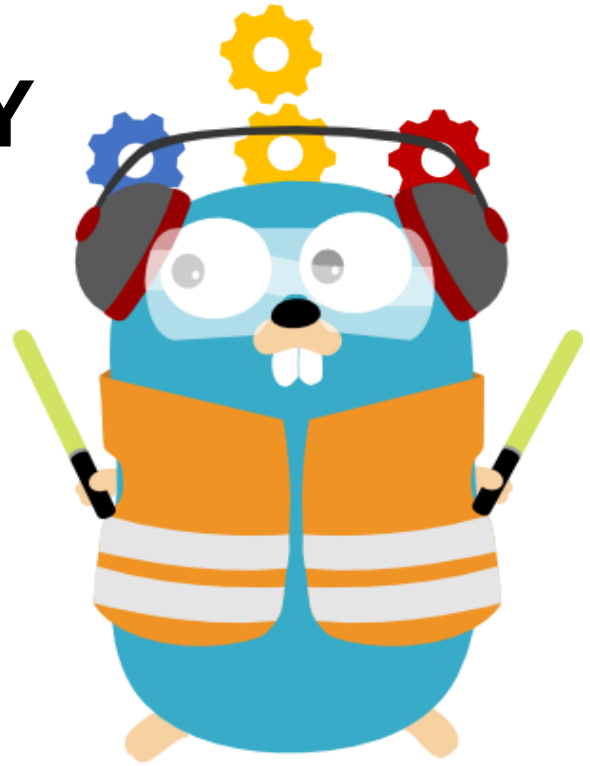
REPOSITORY INSPECTION

UNDOING CHANGES

REWRITING HISTORY

CHITKARA UNIVERSITY

git REPOSITORY



- A directory or storage spece where your projects can live.
- It can be local to folder on your computer, or it can be a storage space on GitHub or another online host.
- You can keep code files, text files, image files, you name it, inside a repository.

Types of repositories

- Central Repository (global)

- Local Repository

CHITKARA UNIVERSITY

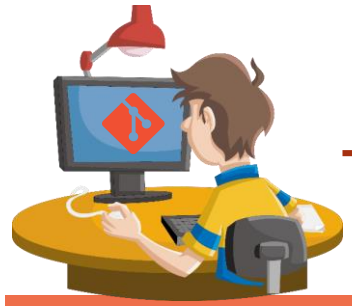
git REPOSITORY

Global Repository

Typically located on remote server

Exclusively consists of “git” repository folder

Meant for team to shear and exchange data

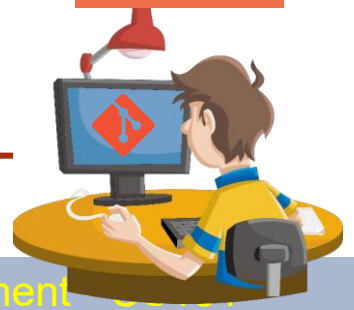


Local Repository

Cenral Repository GitHub

Local Repository

Local Repository



Local Repository

Typically located on the local machine

Resides as a .git folder inside your project’s root

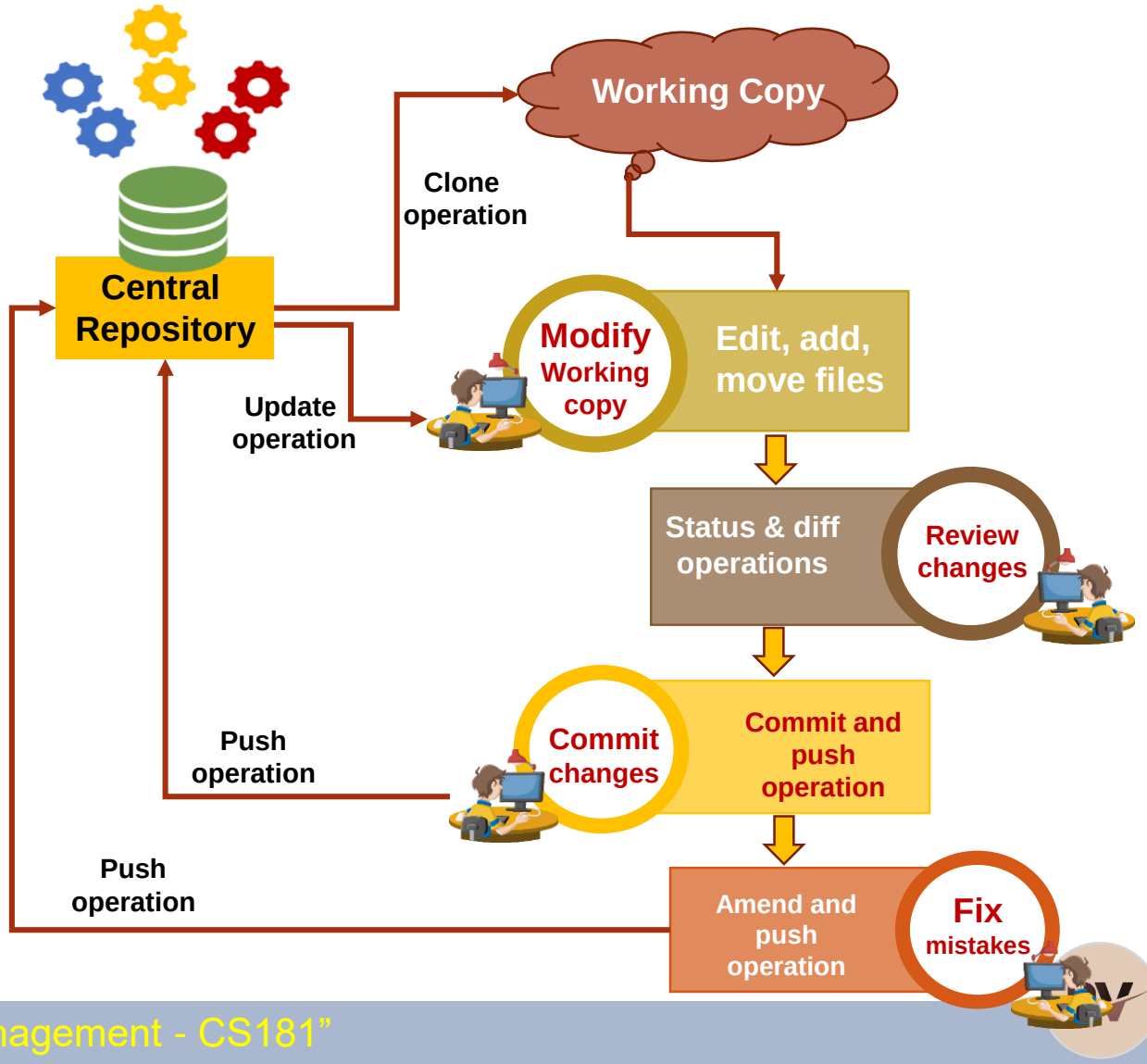
Only admin of the machine can work with this repo

CHITKARA UNIVERSITY

git LIFE CYCLE Work Flow

General workflow is as follows −

1. You clone the Git repository as a working copy.
2. You modify the working copy by adding/editing files.
3. If necessary, you also update the working copy by taking other developer's changes.
4. You review the changes before commit.
5. You commit changes. If everything is fine, then you push the changes to the repository.
6. After committing, if you realize something is wrong, then you correct the last commit and push the changes to the repository.



CHITKARA UNIVERSITY

FILE MANAGEMENT
Work Flow

Untracked

It means that Git sees a file you didn't have in the previous snapshot (commit), and which hasn't yet been staged

Unmodified

"unmodified" means

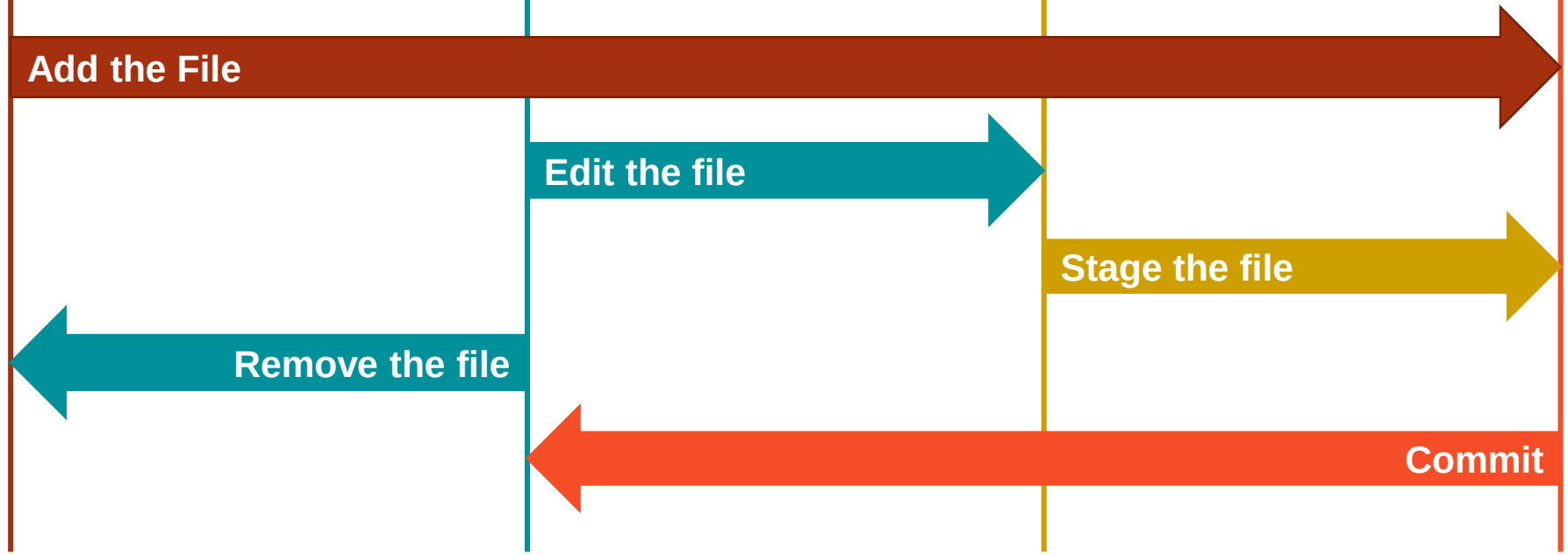
unmodified since the last commit

Modified

As you edit files, Git sees them as modified, because you've changed them since your last commit.

Staged

A staging step in git allows you to continue making changes to the working directory.



CHITKARA UNIVERSITY

git REPOSITORY OPERATIONS

Creating & Setting Up

- Init
- Clone
- Config
- Alias

Syncing

- Add origin
- Push
- Pull

Making Changes

- Stash
- Add
- Commit
- Diff

Parallel Development

- Branch
- Merge
- Rebase

CHITKARA UNIVERSITY

git

REPOSITORY

SETTING UP

An overview of how to set up a repository (repo) under Git version control. We will walk you through initializing a Git repository for a new or existing project. Included below are workflow examples of repositories both created locally and cloned from remote repositories. This guide assumes a basic familiarity with a command-line interface.

init | **clone** | **config** | **alias**

CHITKARA UNIVERSITY

git REPOSITORY init

- To create a new repo, you'll use the git init command. git init is a one-time command you use during the initial setup of a new repo.
- Executing this command will create a new .git subdirectory in your current working directory. This will also create a new main branch.

Versioning an existing project with a new git repository

This example assumes you already have an existing project folder that you would like to create a repo within. You'll first cd to the root project folder and then execute the git init command.

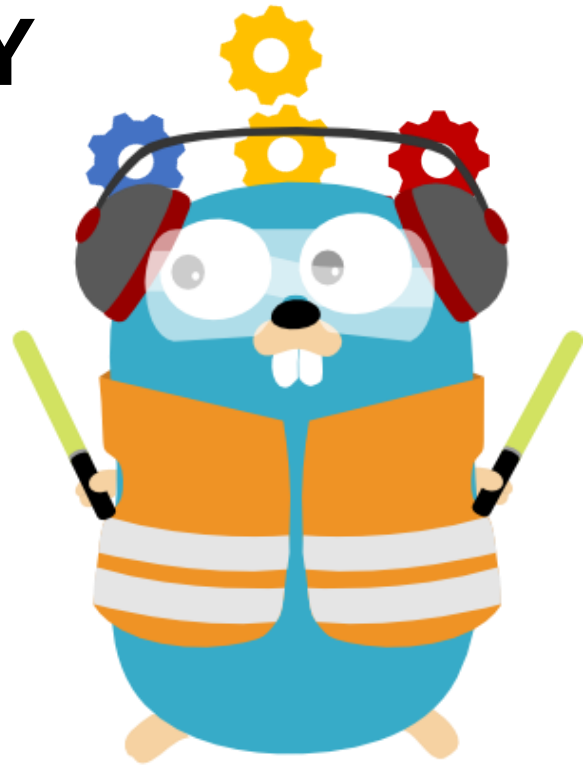
cd /path/to/your/existing/code
git init

Pointing git init to an existing project directory will execute the same initialization setup as mentioned above, but scoped to that project directory.

git init <project directory>

CHITKARA UNIVERSITY

git REPOSITORY clone



- If a project has already been set up in a central repository, the clone command is the most common way for users to obtain a local development clone.
- Like git init, cloning is generally a one-time operation. Once a developer has obtained a working copy, all version control operations are managed through their local repository.

git clone <repo url>

- git clone is used to create a copy or clone of remote repositories. You pass git clone a repository URL.
- Git supports a few different network protocols and corresponding URL formats.
- In this example, we'll be using the Git SSH protocol. Git SSH URLs follow a template of: git@HOSTNAME:USERNAME/REPONAME.git

- When executed, the latest version of the remote repo files on the main branch will be pulled down and added to a new folder.
- The new folder will be named after the REPONAME in this case javascript-data-store. The folder will contain the full history of the remote repository and a newly created main branch.

An example Git SSH URL would be: git@bitbucket.org:rhyolight/javascript-data-store.git where the template values match:

HOSTNAME: bitbucket.org
USERNAME: rhyolight
REPONAME: javascript-data-store

CHITKARA UNIVERSITY

git REPOSITORY config

- Once you have a remote repo setup, you will need to add a remote repo url to your local git config, and set an upstream branch for your local branches. The git remote command offers such utility.

git remote add <remote_name> <remote_repo_url>

- This command will map remote repository at to a ref in your local repo under . Once you have mapped the remote repo you can push local branches to it.

git push -u <remote_name> <local_branch_name>

- This command will push the local repo branch under < local_branch_name > to the remote repo at < remote_name >.

- Git stores configuration options in three separate files, which lets you scope options to individual repositories (local), user (Global), or the entire system (system):

  - Local: /.git/config – Repository-specific settings.
  - Global: /.gitconfig – User-specific settings. This is where options set with the --global flag are stored.
  - System: $(prefix)/etc/gitconfig – System-wide settings.

- git config --global user.name <name>
- git config --local user.email <email>
- git config --global alias.<alias-name> <git-command>
- git config --global alias.ci commit
- git config --system core.editor <editor>
- git config --global --edit

CHITKARA UNIVERSITY

git REPOSITORY alias

- The term alias is synonymous with a shortcut.
- Alias creation is a common pattern found in other popular utilities like `bash` shell.
- Aliases are used to create shorter commands that map to longer commands.
- Aliases enable more efficient workflows by requiring fewer keystrokes to execute a command.
- For a brief example, consider the git checkout command.
- The checkout command is a frequently used git command, which adds up in cumulative keystrokes over time.
- An alias can be created that maps git co to git checkout, which saves precious human fingertip power by allowing the shorter keystroke form: **git co** to be typed instead.

- It is important to note that there is no direct git alias command.
- Aliases are created through the use of the git config command and the Git configuration files. As with other configuration values, aliases can be created in a local or global scope.

To better understand Git aliases let us create some examples.

- $ git config --global alias.co checkout
- $ git config --global alias.br branch
- $ git config --global alias.ci commit
- $ git config --global alias.st status

```
[alias]
        co = checkout
        br = branch
        ci = commit
        st = status
```

- This code example creates globally stored shortcuts for common git commands.
- Creating the aliases will not modify the source commands. So git checkout will still be available even though we now have the git co alias.
- These aliases were created with the --global flag which means they will be stored in Git's global operating system level configuration file.
- On linux systems, the global config file is located in the User home directory at /.gitconfig.

This demonstrates that the aliases are now equivalent to the source commands.

CHITKARA UNIVERSITY

git

REPOSITORY

SAVING CHANGES

When working in Git, or other version control systems, the concept of "saving" is a more nuanced process than saving in a word processor or other traditional file editing applications. The traditional software expression of "saving" is synonymous with the Git term "committing". A commit is the Git equivalent of a "save".

add | **commit** | **diff** | **stash & .gitignore**

CHITKARA UNIVERSITY

git REPOSITORY add

- The git add command adds a change in the working directory to the staging area.
- It tells Git that you want to include updates to a particular file in the next commit.
- However, git add doesn't really affect the repository in any significant way—changes are not actually recorded until you run git commit.

Common options

- Stage all changes in <file> for the next commit. **git add <file>**
- Stage all changes in <directory> for the next commit. **git add <directory>**
- Begin an interactive staging session that lets you choose portions of a file to add to the next commit. **git add -p**
- This will present you with a chunk of changes and prompt you for a command. Use y to stage the chunk, n to ignore the chunk, s to split it into smaller chunks, e to manually edit the chunk, and q to exit.

CHITKARA UNIVERSITY

git REPOSITORY commit

- The git commit command captures a snapshot of the project's currently staged changes.
- Committed snapshots can be thought of as “safe” versions of a project—Git will never change them unless you explicitly ask it to.
- Prior to the execution of git commit, The git add command is used to promote or 'stage' changes to the project that will be stored in a commit.
- These two commands git commit and git add are two of the most frequently used.

Common options

- Commit the staged snapshot. This will launch a text editor prompting you for a commit message.
- After you’ve entered a message, save the file and close the editor to create the actual commit.

git commit

- Commit a snapshot of all changes in the working directory.
- This only includes modifications to tracked files (those that have been added with git add at some point in their history).

git commit -a

- A shortcut command that immediately creates a commit with a passed commit message.
- By default, git commit will open up the locally configured text editor, and prompt for a commit message to be entered.
- Passing the -m option will forgo the text editor prompt in-favor of an inline message.

git commit -m "commit message“

- A power user shortcut command that combines the -a and -m options. This combination immediately creates a commit of all the staged changes and takes an inline commit message.

git commit -am "commit message"

- This option adds another level of functionality to the commit command. Passing this option will modify the last commit. Instead of creating a new commit, staged changes will be added to the previous commit.
- This command will open up the system's configured text editor and prompt to change the previously specified commit message.

git commit --amend

CHITKARA UNIVERSITY

git REPOSITORY diff

git-diff - Show changes between commits, commit and working tree, etc

- Diffing is a function that takes two input data sets and outputs the changes between them.
- git diff is a multi-use Git command that when executed runs a diff function on Git data sources.
- These data sources can be commits, branches, files and more.
- The git diff command is often used along with git status and git log to analyze the current state of a Git repo.

Git diff: common commands

- git diff [<options>] [<commit>] [--] [<path>…]
- git diff [<options>] --cached [--merge-base] [<commit>] [--] [<path>…]
- git diff [<options>] [--merge-base] <commit> [<commit>…] <commit> [--] [<path>…]
- git diff [<options>] <commit>…<commit> [--] [<path>…]
- git diff [<options>] <blob> <blob>
- git diff [<options>] --no-index [--] <path> <path>

Options

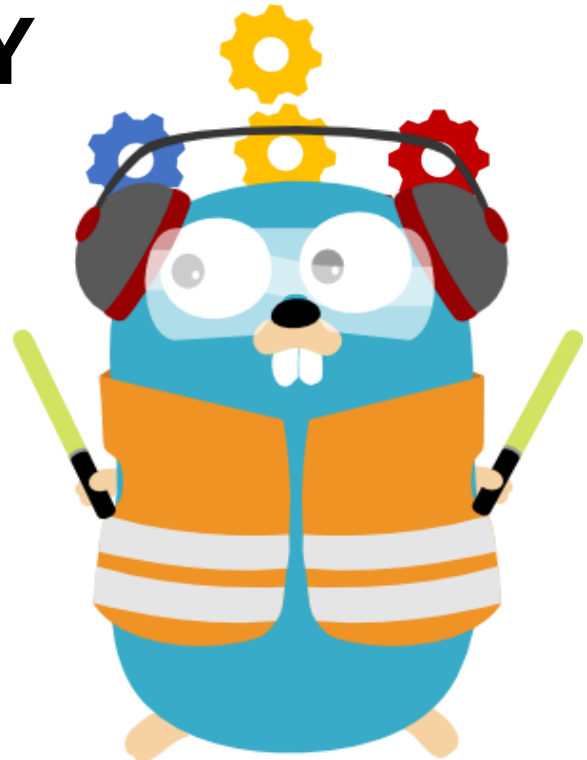
-p
-u
--patch
Generate patch (see section on generating patches). This is the default.

-s
--no-patch
Suppress diff output. Useful for commands like git show that show the patch by default, or to cancel the effect of --patch.

-U<n>
--unified=<n>
Generate diffs with <n> lines of context instead of the usual three. Implies --patch.

CHITKARA UNIVERSITY

git REPOSITORY diff



- Diffing is a function that takes two input data sets and outputs the changes between them.
- git diff is a multi-use Git command that when executed runs a diff function on Git data sources.
- These data sources can be commits, branches, files and more.
- This document will discuss common invocations of git diff and diffing work flow patterns.
- The git diff command is often used along with git status and git log to analyze the current state of a Git repo.

- By default git diff will show you any uncommitted changes since the last commit.

git diff

Common options

- **Comparison input:** This line displays the input sources of the diff. We can see that a/diff_test.txt and b/diff_test.txt have been passed to the diff.

diff --git a/diff_test.txt b/diff_test.txt

- git diff also has a special mode for highlighting changes with much better granularity: --color-words. This mode tokenizes added and removed lines by whitespace and then diffs those.

git diff --color-words

- Diff-highlight pairs up matching lines of diff output and highlights sub-word fragments that have changed.

git diff-highlight

- When git diff is invoked with the --cached option the diff will compare the staged changes with the local repository. The --cached option is synonymous with --staged.

git diff --cached ./path/to/file

- **Comparing two branches:** Branches are compared like all other ref inputs to git diff. This example introduces the dot operator. The two dots in this example indicate the diff input is the tips of both branches.

git diff branch1..other-feature-branch

- To compare a specific file across branches, pass in the path of the file as the third argument to git diff

git diff main new_branch ./diff_test.txt

CHITKARA UNIVERSITY

git REPOSITORY stash

git stash: operation takes your modified tracked files, stages changes, and saves them on a stack of unfinished changes that you can reapply at any time.

Use Case

- Suppose you are implementing a new feature for your product. Your code is in progress and suddenly a customer escalation comes. Because of this, you have to keep aside your new feature work for a few hours. You cannot commit your partial code and also cannot throw away your changes. So you need some temporary space, where you can store your partial changes and later on commit it.

- git stash temporarily shelves (or stashes) changes you've made to your working copy so you can work on something else, and then come back and re-apply them later on.
- Stashing is handy if you need to quickly switch context and work on something else, but you're mid-way through a code change and aren't quite ready to commit.

Git stash: common commands

```
$ git status -s
$ git stash
$ git stash list
```

CHITKARA UNIVERSITY

git REPOSITORY stash

- git stash temporarily shelves (or stashes) changes user's made to their working copy so user can work on something else, and then come back and re-apply them later on.
- Stashing is handy if user need to quickly switch context and work on something else, but user mid-way through a code change and aren't quite ready to commit.
- 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.

Common options

- It stores (or "stashes") the uncommitted changes (staged and unstaged files) and overlooks untracked and ignored files.

git status

- Stash untracked files

git stash -u (or)
git stash --include-untracked

- Stash untracked files and ignored files.

git stash -a (or)
git stash --all

- Stashes are saved in a last-in-first-out (LIFO) approach.

git stash list

- Empties the stash list by removing all the stashes.

git stash clear

- Deletes a particular stash from the stash list.

git stash drop <stash_id>

- You can choose which stash you want to pop or apply by passing the identifier as the last argument:

git stash pop stash@{1} (or)
git stash apply stash@{1}

CHITKARA UNIVERSITY

git REPOSITORY .gitignore

- Git sees every file in your working copy as one of three things:
  - **tracked -** a file which has been previously staged or committed;
  - **untracked -** a file which has not been staged or committed; or
  - **ignored -** a file which Git has been explicitly told to ignore.
- Ignored files are usually build artifacts and machine generated files that can be derived from your repository source or should otherwise not be committed. Some common examples are:
  - dependency caches, such as the contents of /node_modules or /packages
  - compiled code, such as .o, .pyc, and .class files
  - build output directories, such as /bin, /out, or /target
  - files generated at runtime, such as .log, .lock, or .tmp
  - hidden system files, such as .DS_Store or Thumbs.db
  - personal IDE config files, such as .idea/workspace.xml
- Ignored files are tracked in a special file named .gitignore that is checked in at the root of your repository.
- There is no explicit git ignore command: instead the .gitignore file must be edited and committed by hand when you have new files that you wish to ignore. .gitignore files contain patterns that are matched against file names in your repository to determine whether or not they should be ignored.

CHITKARA UNIVERSITY

git

REPOSITORY PARALLEL DEVELOPMENT

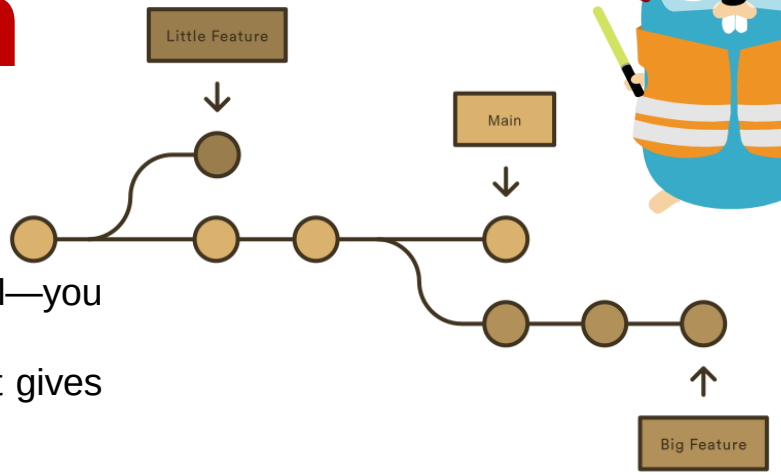
When working in Git, or other version control systems, the concept of "saving" is a more nuanced process than saving in a word processor or other traditional file editing applications. The traditional software expression of "saving" is synonymous with the Git term "committing". A commit is the Git equivalent of a "save".

branch | checkout | merge | conflicts

CHITKARA UNIVERSITY

git REPOSITORY branch

- Branching is a feature available in most modern version control systems.
- Branching in other VCS's can be an expensive operation in both time and disk space.
- Git branches are effectively a pointer to a snapshot of your changes.
- When you want to add a new feature or fix a bug—no matter how big or how small—you spawn a new branch to encapsulate your changes.
- This makes it harder for unstable code to get merged into the main code base, and it gives you the chance to clean up your future's history before merging it into the main branch.



Common options

- List all of the branches in your repository. This is synonymous with git branch --list. **git branch**
- Create a new branch called <branch>. This does not check out the new branch. **git branch <branch>**
- Rename the current branch to <branch>. **git branch -m <branch>**
- Delete the specified branch. This is a “safe” operation in that Git prevents you from deleting the branch if it has unmerged changes. **git branch -d <branch>**
- Force delete the specified branch, even if it has unmerged changes. This is the command to use if you want to permanently throw away all of the commits associated with a particular line of development. **git branch -D <branch>**
- List all remote branches. **git branch -a**

CHITKARA UNIVERSITY

git REPOSITORY checkout

- In Git terms, a "checkout" is the act of switching between different versions of a target entity.
- The git checkout command operates upon three distinct entities: files, commits, and branches.
- In addition to the definition of "checkout" the phrase "checking out" is commonly used to imply the act of executing the git checkout command.

Common options

- The git checkout command accepts a -b argument that acts as a convenience method which will create the new branch and immediately switch to it.
- <existing-branch> is passed which then bases new-branch off of existing-branch instead of the current HEAD.
- Switching branches is a straightforward operation.
- In order to checkout a remote branch you have to first fetch the contents of the branch.
- In modern versions of Git, you can then checkout the remote branch like a local branch.
- Older versions of Git require the creation of a new branch based on the remote.
- Additionally you can checkout a new local branch and reset it to the remote branches last commit.

git checkout -b <new-branch>

git checkout -b <new-branch> <existing-branch>

git checkout <branchname>

git fetch --all

git checkout <remotebranch>

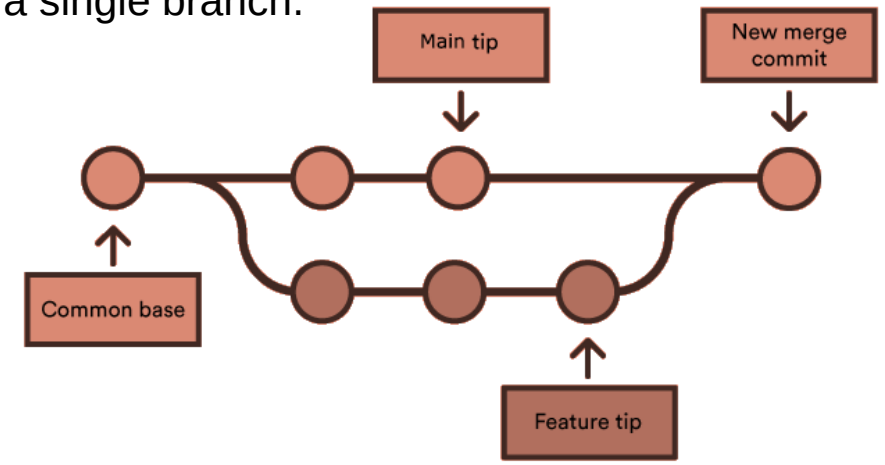
git checkout -b <remotebranch> origin/<remotebranch>

git checkout -b <branchname>
git reset --hard origin/<branchname>

CHITKARA UNIVERSITY

git REPOSITORY merge

- Merging is Git's way of putting a forked history back together again.
- The git merge command lets you take the independent lines of development created by git branch and integrate them into a single branch.
- git merge is used to combine two branches.



Common options

- The branch that will be merged into the receiving branch.

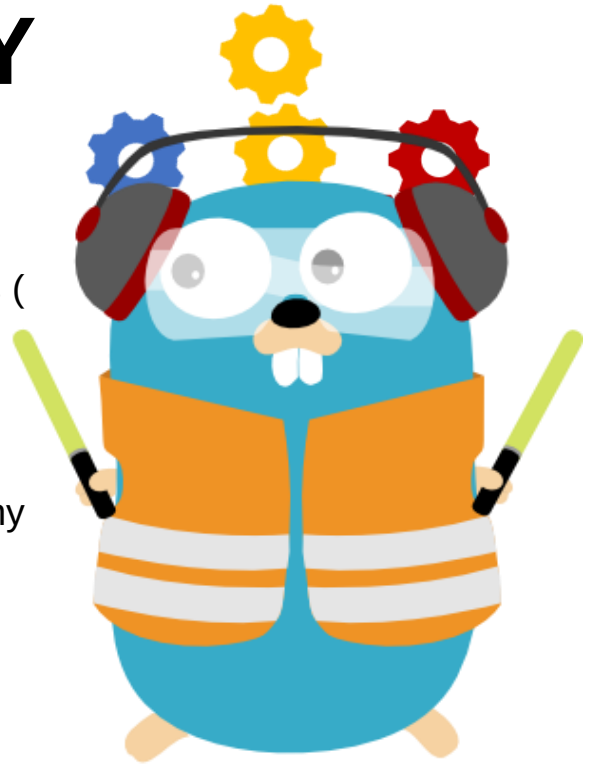
git merge <branch_name>

- In the event that user require a merge commit during a fast forward merge for record keeping purposes user to execute git merge with the --no-ffoption.

git merge --no-ff <branch>

CHITKARA UNIVERSITY

git REPOSITORY conflicts



- Version control systems are all about managing contributions between multiple distributed authors (usually developers).
- Sometimes multiple developers may try to edit the same content.
- If Developer A tries to edit code that Developer B is editing a conflict may occur.
- To alleviate the occurrence of conflicts developers will work in separate isolated branches.
- The git merge command's primary responsibility is to combine separate branches and resolve any conflicting edits.

Common options

Git commands that can help resolve merge conflicts

- The status command is in frequent use when a working with Git and during a merge it will help identify conflicted files. **git status**
- Passing the --merge argument to the git log command will produce a log with a list of commits that conflict between the merging branches. **git log --merge**
- diff helps find differences between states of a repository/files. This is useful in predicting and preventing merge conflicts. **git diff**
- checkout can be used for undoing changes to files, or for changing branches **git checkout**
- reset can be used to undo changes to the working directory and staging area. **git reset --mixed**
- Executing git merge with the --abort option will exit from the merge process and return the branch to the state before the merge began. **git merge --abort**
- Git reset can be used during a merge conflict to reset conflicted files to a know good state **git reset**

CHITKARA UNIVERSITY

git

REPOSITORY

Collaborating

Syncing

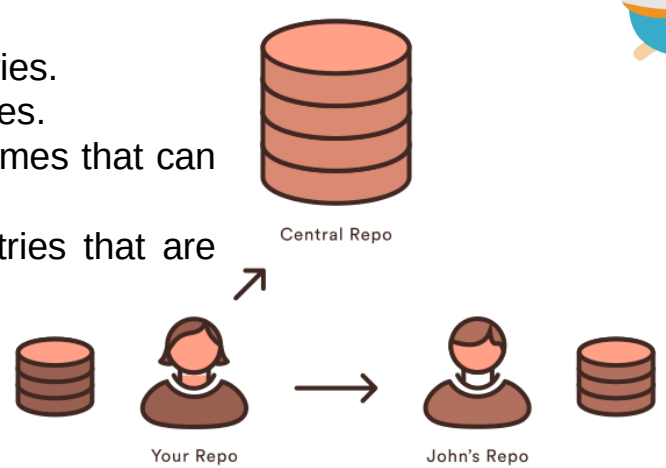
When working in Git, or other version control systems, the concept of "saving" is a more nuanced process than saving in a word processor or other traditional file editing applications. The traditional software expression of "saving" is synonymous with the Git term "committing". A commit is the Git equivalent of a "save".

remote | fetch | push | pull

CHITKARA UNIVERSITY

git REPOSITORY remote

- The git remote command lets user create, view, and delete connections to other repositories.
- Remote connections are more like bookmarks rather than direct links into other repositories.
- Instead of providing real-time access to another repository, they serve as convenient names that can be used to reference a not-so-convenient URL.
- The git remote command is essentially an interface for managing a list of remote entries that are stored in the repository's ./.git/config file.

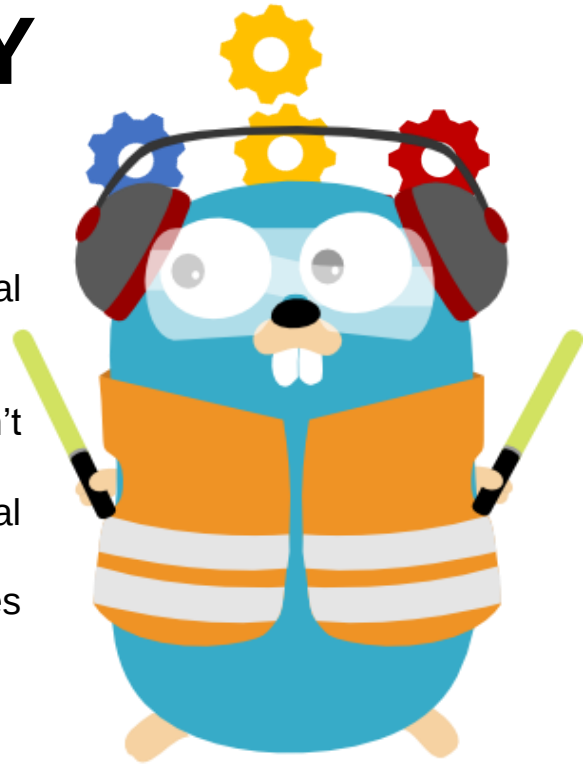


Common options

• List the remote connections you have to other repositories.	**git remote**
• Same as the above command, but include the URL of each connection.	**git remote -v**
• Create a new connection to a remote repository. After adding a remote, you'll be able to use ＜name＞ as a convenient shortcut for ＜url＞ in other Git commands.	**git remote add <name> <url>**
• Remove the connection to the remote repository called ＜name＞.	**git remote rm <name>**
• Rename a remote connection from ＜old-name＞ to ＜new-name＞.	**git remote rename <old-name> <new-name>**
• Create a new connection to a remote repository. After adding a remote, you'll be able to use ＜name＞ as a convenient shortcut for ＜url＞ in other Git commands.	**git remote add <name> <url>**

CHITKARA UNIVERSITY

git REPOSITORY fetch



- The git fetch command downloads commits, files, and refs from a remote repository into user local repo.
- Fetching is what you do when you want to see what everybody else has been working on.
- It's similar to svn update in that it lets you see how the central history has progressed, but it doesn't force user to actually merge the changes into your repository.
- Git isolates fetched content from existing local content; it has absolutely no effect on your local development work.
- Fetched content has to be explicitly checked out using the git checkout command. This makes fetching a safe way to review commits before integrating them with your local repository.

Common options

- Fetch all of the branches from the repository. This also downloads all of the required commits and files from the other repository.

git fetch <remote>

- Same as the above command, but only fetch the specified branch.

git fetch <remote> <branch>

- A power move which fetches all registered remotes and their branches:

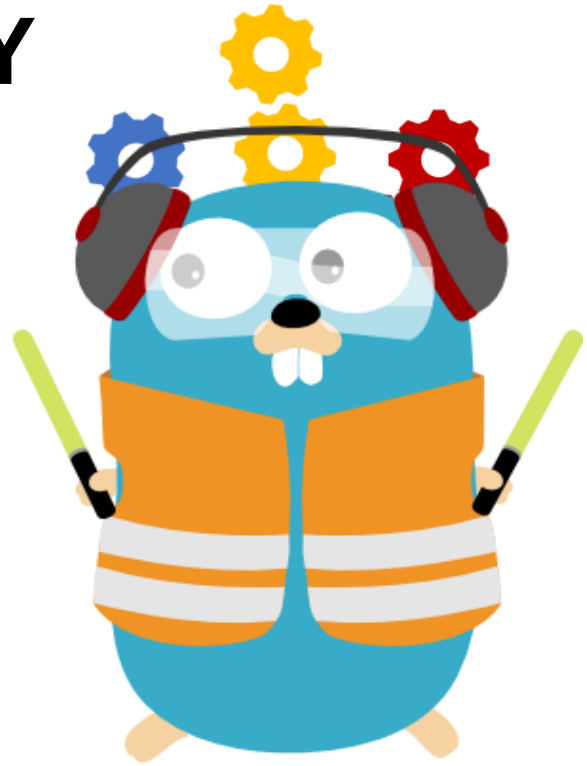
git fetch --all

- The --dry-run option will perform a demo run of the command. It will output examples of actions it will take during the fetch but not apply them.

git fetch --dry-run

CHITKARA UNIVERSITY

git REPOSITORY push



- The git push command is used to upload local repository content to a remote repository.
- Pushing is how you transfer commits from your local repository to a remote repo.
- It's the counterpart to git fetch, but whereas fetching imports commits to local branches, pushing exports commits to remote branches.
- Remote branches are configured using the git remote command.
- Pushing has the potential to overwrite changes, caution should be taken when pushing.

Common options

- Push the specified branch to , along with all of the necessary commits and internal objects. This creates a local branch in the destination repository. To prevent you from overwriting commits, Git won't let you push when it results in a non-fast-forward merge in the destination repository.

git push <remote> <branch>

- Same as the above command, but force the push even if it results in a non-fast-forward merge. Do not use the --force flag unless you're absolutely sure you know what you're doing.

git push <remote> --force

- Push all of your local branches to the specified remote.

git push <remote> --all

- Tags are not automatically pushed when you push a branch or use the --all option. The --tags flag sends all of your local tags to the remote repository.

git push <remote> --tags

CHITKARA UNIVERSITY

git REPOSITORY pull

- The git pull command is used to fetch and download content from a remote repository and immediately update the local repository to match that content.
- Merging remote upstream changes into your local repository is a common task in Git-based collaboration work flows.
- The git pull command is actually a combination of two other commands, git fetch followed by git merge.
- In the first stage of operation git pull will execute a git fetch scoped to the local branch that HEAD is pointed at. Once the content is downloaded, git pull will enter a merge workflow.
- A new merge commit will be-created and HEAD updated to point at the new commit.

Common options

- Fetch the specified remote's copy of the current branch and immediately merge it into the local copy. This is the same as git fetch ＜remote＞ followed by git merge origin/＜current-branch＞.

git pull <remote>

- Similar to the default invocation, fetches the remote content but does not create a new merge commit.

git pull --no-commit <remote>

- Same as the previous pull Instead of using git merge to integrate the remote branch with the local one, use git rebase.

git pull --rebase <remote>

- Gives verbose output during a pull which displays the content being downloaded and the merge details.

git pull --verbose

CHITKARA UNIVERSITY

git

Q & A