Chapter 03
GIT Basics

Open Source SW Development
CSE22300

Version Control

Problems Working Alone

- **Ever done one of the following?**
  - **Had code that worked, made a bunch of changes and saved it, which broke the code, and now you just want the working version back…**
  - **Accidentally deleted a critical file, hundreds of lines of code gone…**
  - **Somehow messed up the structure/contents of your code base, and want to just "undo" the crazy action you just did**
  - **Hard drive crash!!!! Everything's gone, the day before deadline.**

- **Possible options:**
  - **Save as (MyClass-v1.java)**
  - **Ugh. Just ugh. And now a single line change results in duplicating the entire file…**

Problems Working in teams

- **Whose computer stores the "official" copy of the project?**
  - **Can we store the project files in a neutral "official" location?**
- **Will we be able to read/write each other's changes?**
  - **Do we have the right file permissions?**
  - **Lets just email changed files back and forth! Yay!**
- **What happens if we both try to edit the same file?**
  - **Bill just overwrote a file I worked on for 6 hours!**
- **What happens if we make a mistake and corrupt an important file?**
  - **Is there a way to keep backups of our project files?**
- **How do I know what code each teammate is working on?**

Dealing with Changes

- **How do you manage your project?**
  - **Modifying existing code**
  - **Backing up working code**
  - **Checking if an idea works (Do I use a Hashtable or a HashMap?)**
  - **Sharing code in group projects**

(Bad) Solution

- **Copying source codes**
  - **project_working.java, project_tmp.java**
- **Copy & Paste code snippets**
- **Copy entire directories**
- **Emailing code to people**

Open Source

- **You thought coursework was bad?**
- **Linux kernel has thousands of regular developers, millions of files.**
- **Developers spread over the globe across multiple time zones**

Big Code Bases

- **Operating systems code**
  - **Win 95 approx 5 million lines of code (1995)**
  - **Linux kernel 2.6.37 14 million lines of code (2011)**

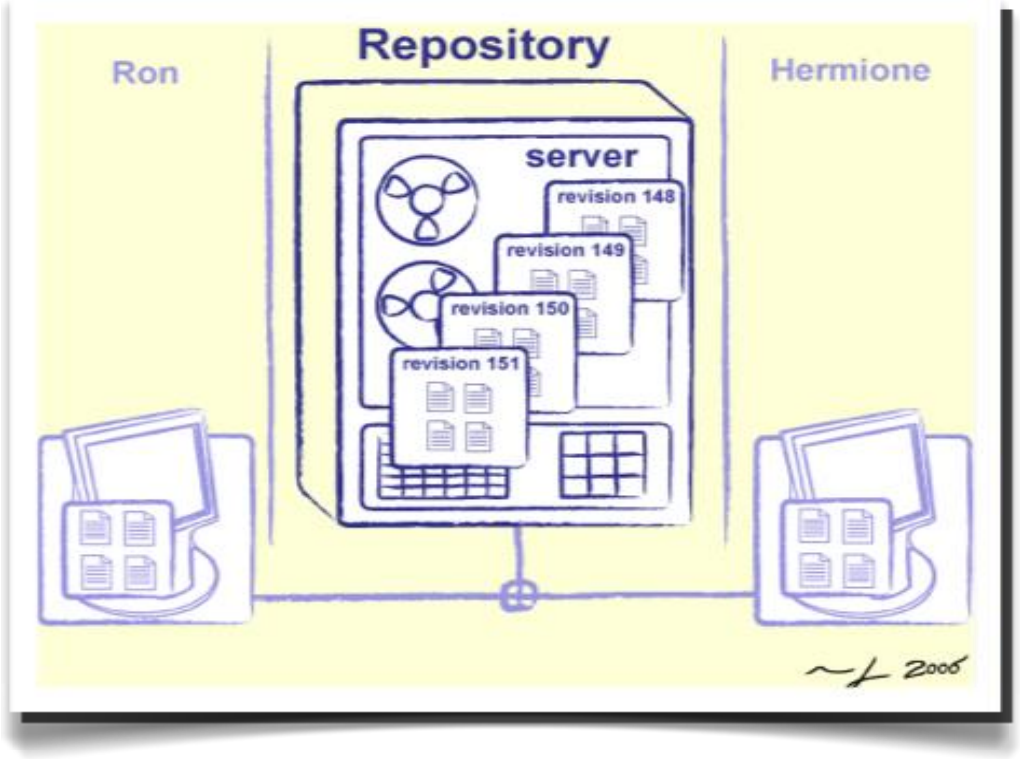
- **Modern PC game**
  - **Unreal 3 approx 500,000 lines of code**

Making a Mess

- **The Linux kernel runs on different processors (ARM, x86, MIPS). These can require significant differences in low level parts of the code base**
- **Many different modules**
- **Old versions are required for legacy systems**
- **Because it is open source, any one can download and suggest changes.**
- **How can we create a single kernel from all of this?**

Control the Process Automatically

- **Manage these things using a version control system (VCS)**
- **A version control system is a system which allows for the management of a code base.**



Repositories

- **Repository (aka “repo”): a location storing a copy of all files.**
  - **you don't edit files directly in the repo;**
  - **you edit a local working copy or “working tree”**
  - **then you commit your edited files into the repo**

- **There may be only one repository that all users share (CVS, Subversion)**
- **Or each user could also have their own copy of the repository (Git, Mercurial)**

- **Files in your working directory must be added to the repo in order to be tracked.**

What to put in a Repo?

- **Everything needed to create your project:**
- **Source code (Examples: .java, .c, .h, .cpp)**
- **Build files (Makefile, build.xml)**
- **Other resources needed to build your project: icons, text etc.**

- **Things generally NOT put in a repo (these can be easily re-created and just take up space):**
- **Object files (.o)**
- **Executables (.exe)**

Repository Location

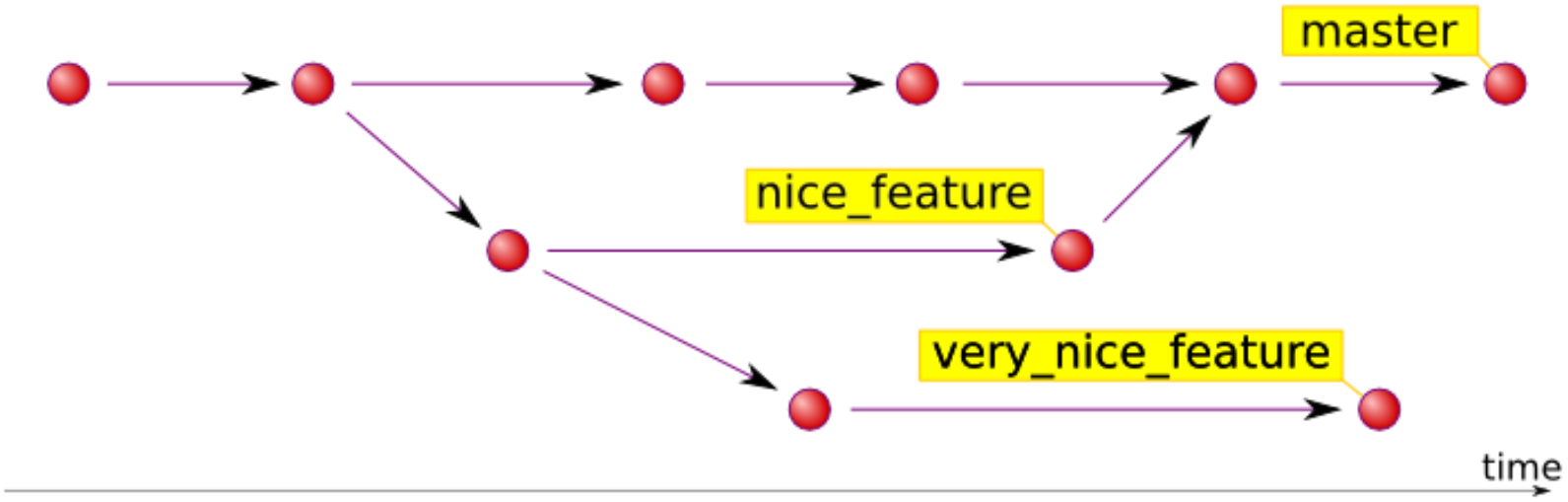
- **Can create the repository anywhere**
  - **Can be on the same computer that you're going to work on, which might be ok for a personal project where you just want rollback protection**

- **But, usually you want the repository to be robust:**
  - **On a computer that's up and running 24/7**
  - **Everyone always has access to the project**
  - **On a computer that has a redundant file system (ie RAID)**
  - **No more worries about that hard disk crash wiping away your project!**

Details of the Process

- **Files are kept in a repository**
- **Repositories can be local or remote to the user**
- **The user edits a copy called the working copy**
- **Changes are committed to the repository when the user is finished making changes**
- **Other people can then access the repository to get the new code**
- **Can also be used to manage files when working across multiple computers**

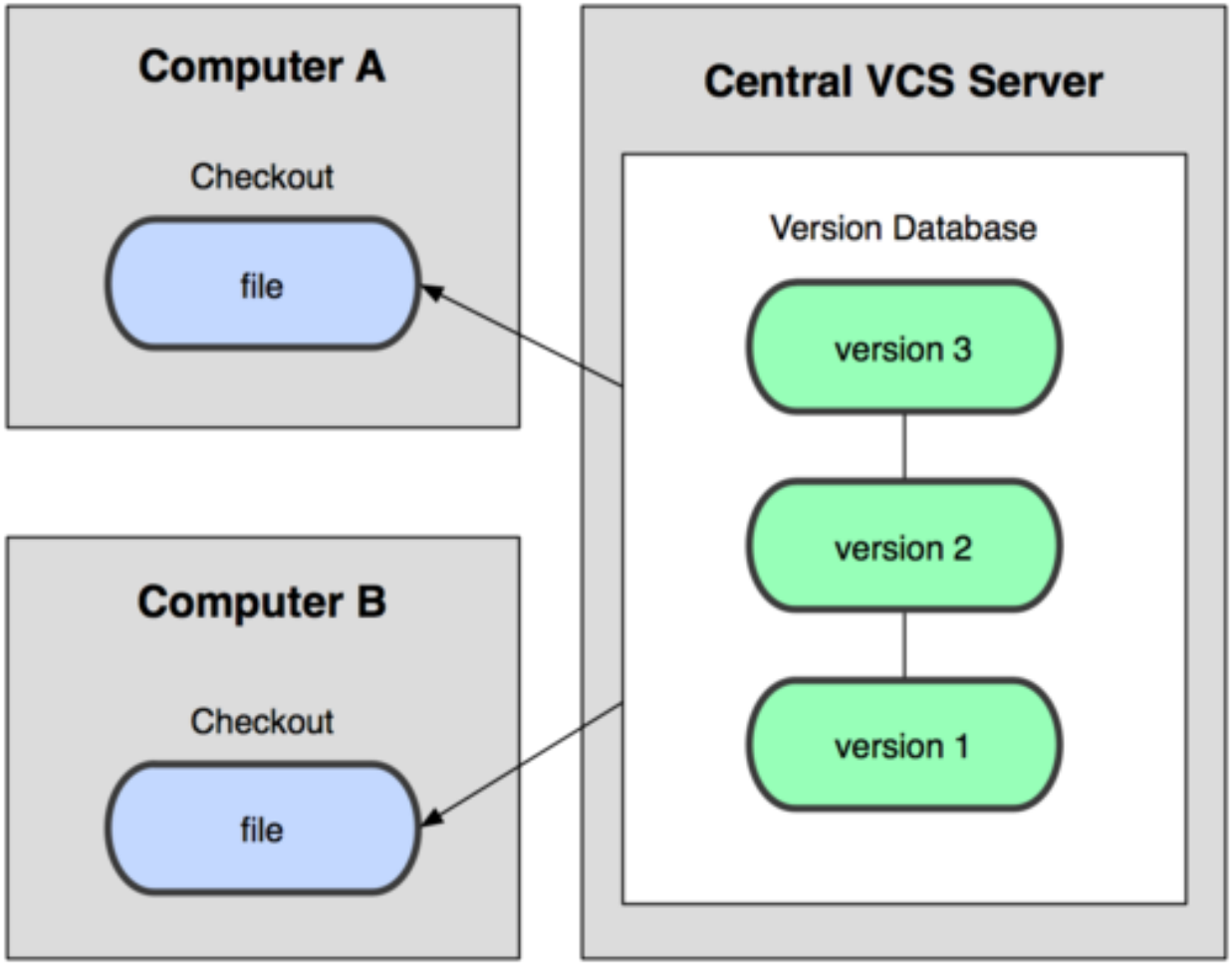
Details of the Process



Centralized Version Control

- **A single server holds the code base**
- **Clients access the server by means of check-in/check-outs**
- **Examples include CVS, Subversion, Visual Source Safe.**
- **Advantages: Easier to maintain a single server.**
- **Disadvantages: Single point of failure.**

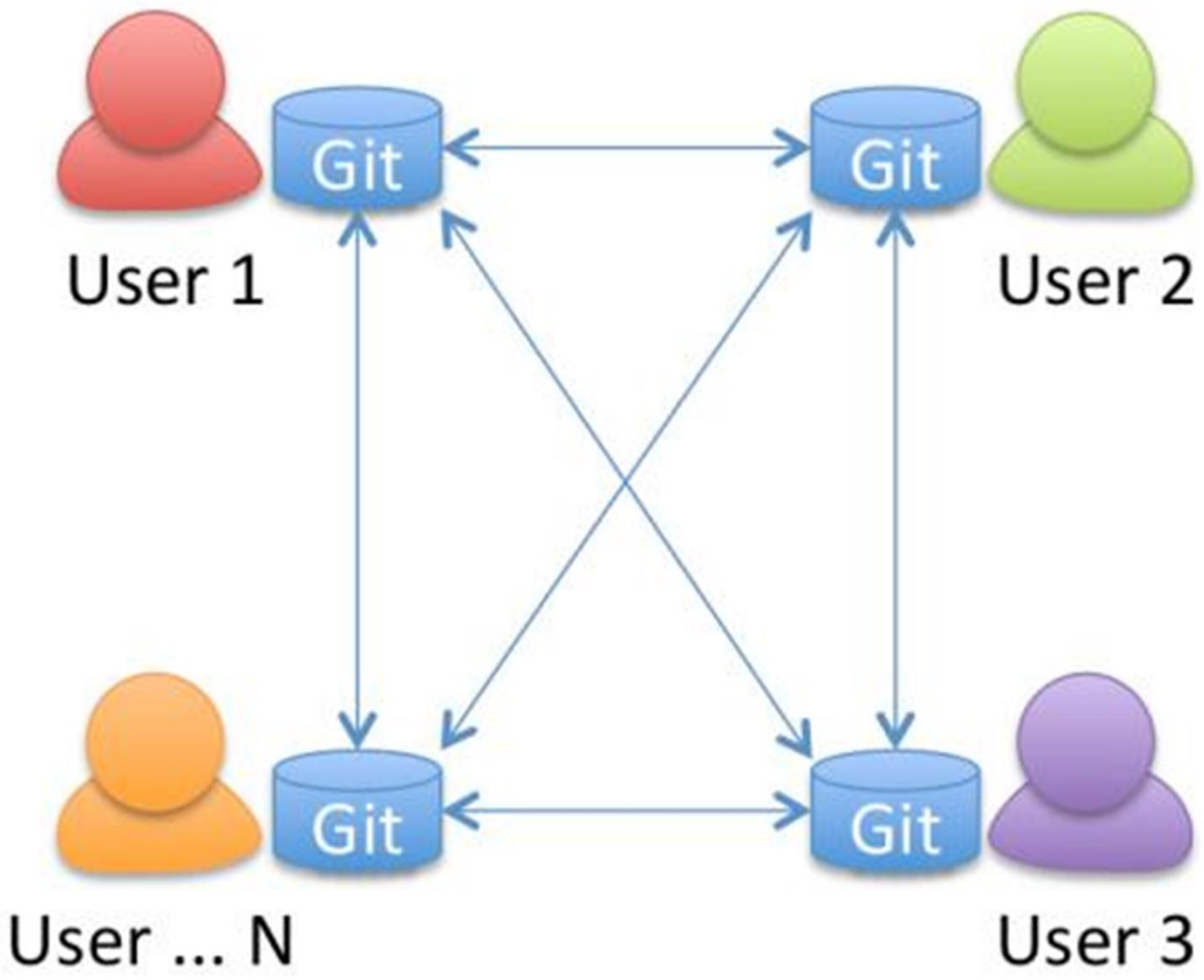
Centralized Version Control



Distributed Version Control

- **Each client (essentially) holds a complete copy of the code base.**
- **Code is shared between clients by push/pulls**
- **Advantages: Many operations cheaper. No single point of failure**
- **Disadvantages: A bit more complicated!**

Distributed Version Control



More Uses of Version Control

- **Version control is not just useful for collaborative working, essential for quality source code development**
- **Often want to undo changes to a file**
  - **start work, realize it's the wrong approach, want to get back to starting point**
  - **like "undo" in an editor…**
  - **keep the whole history of every file and a changelog**
- **Also want to be able to see who changed what, when**
  - **The best way to find out how something works is often to ask the person who wrote it**

Branching

- **Branches allows multiple copies of the code base within a single repository.**
  - **Different customers have different requirements**
  - **Customer A wants features A,B, C**
  - **Customer B wants features A & C but not B because his computer is old and it slows down too much.**
  - **Customer C wants only feature A due to costs**
  - **Each customer has their own branch.**
- **Different versions can easily be maintained**

Selecting a VCS

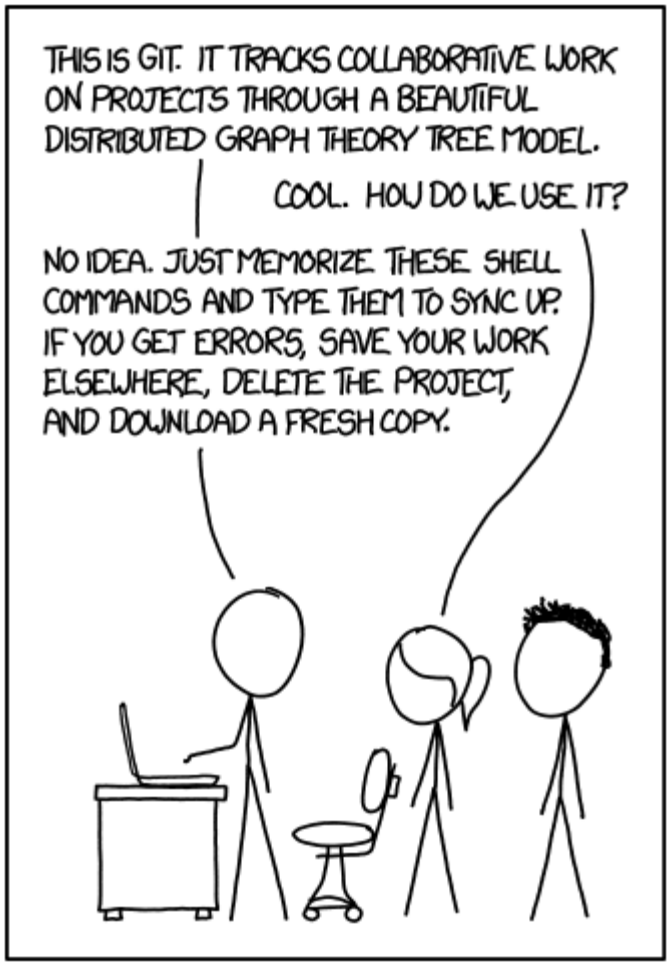
- **When choosing a VCS consider:**
  - **How many files and developers are likely to be involved in the project?**
  - **Speed for common operations (check-in, check-out)**
  - **Is there a server? Does it need to be powerful?**

Essential features

- **Check-in and check-out of items to repository**
- **Creation of baselines (labels/tags)**
  - **Version 1.0 released!**
- **Control and manipulation of branching**
  - **management of multiple versions**
- **Overview of version history**

GIT

Git



HTTP://XKCD.COM/1597/

History of Git

- **Came out of Linux development community**
- **Linus Torvalds, 2005**
- **Initial goals:**
  - **Speed**
  - **Support for non-linear development (thousands of parallel branches)**
  - **Fully distributed**
  - **Able to handle large projects like Linux efficiently**

Git Advantages

- **Resilience**
  - **No one repository has more data than any other**
- **Speed**
  - **Very fast operations compared to other VCS (I'm looking at you CVS and Subversion)**
- **Space**
  - **Compression can be done across repository not just per file**
  - **Minimizes local size as well as push/pull data transfers**
- **Simplicity**
  - **Object model is very simple**
- **Large userbase with robust tools**

Git Disadvantages

- **Definite learning curve, especially for those used to centralized systems**
  - **Can sometimes seem overwhelming to learn**
  - **Conceptual difference**
  - **Huge amount of commends**

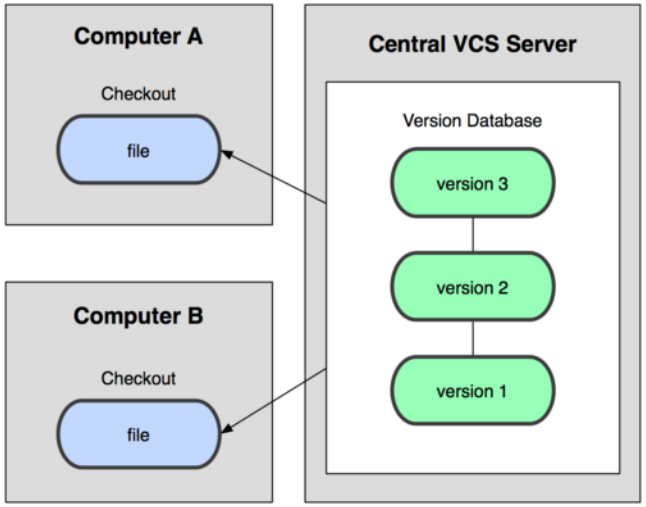
Git Resources

- **At the command line: (where <verb> = config, add, commit)**
  - **$ git help <verb>**
  - **$ git <verb> --help**
  - **$ man git-<verb>**

- **Free on-line book: https://git-scm.com/book/en/v2**
- **Git tutorial: http://schacon.github.com/git/gittutorial.html**
- **Reference page for Git: http://gitref.org/index.html**
- **Git website: http://git-scm.com/**

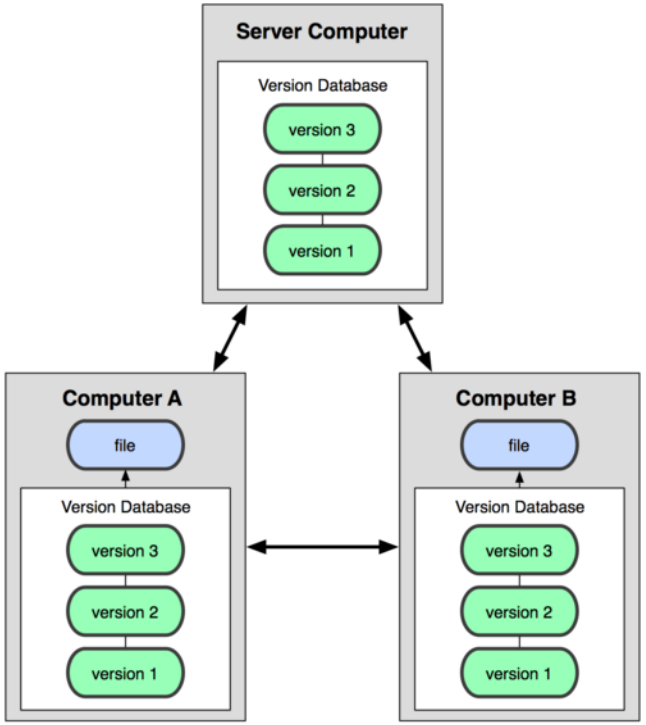
Git uses a distributed model

Centralized Model



(CVS, Subversion, Perforce)

Distributed Model



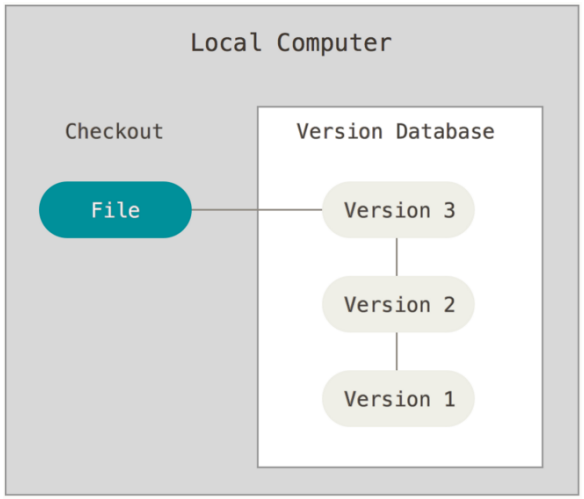
(**Git**, Mercurial)

Result: Many operations are local

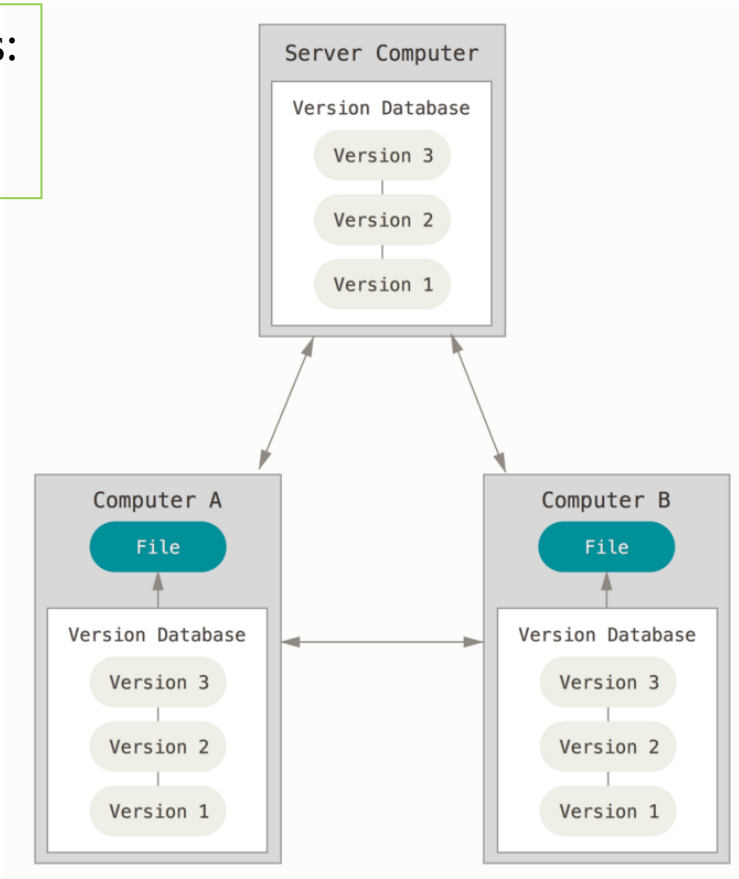
Ways to use Git

Possible servers:

- Git Server
- GitHub

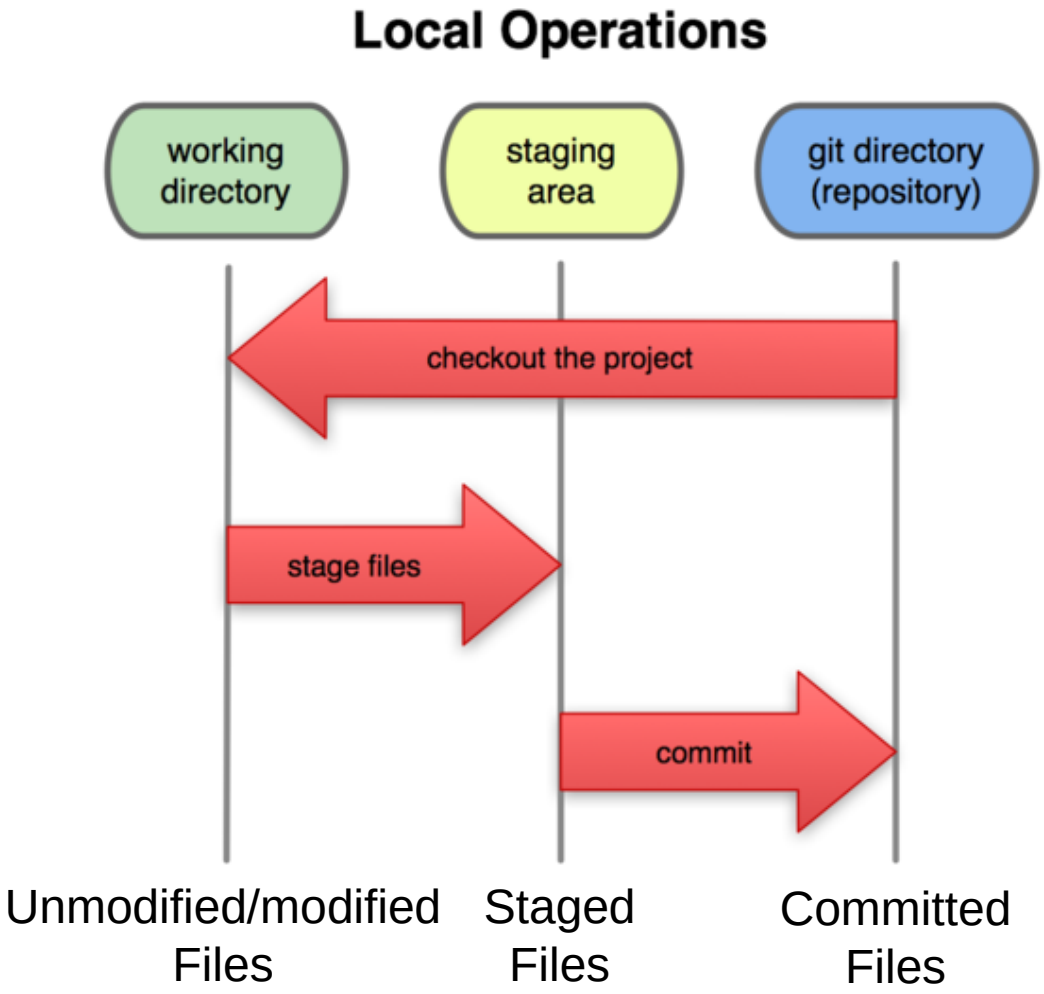


Using Git on your own computer, one user



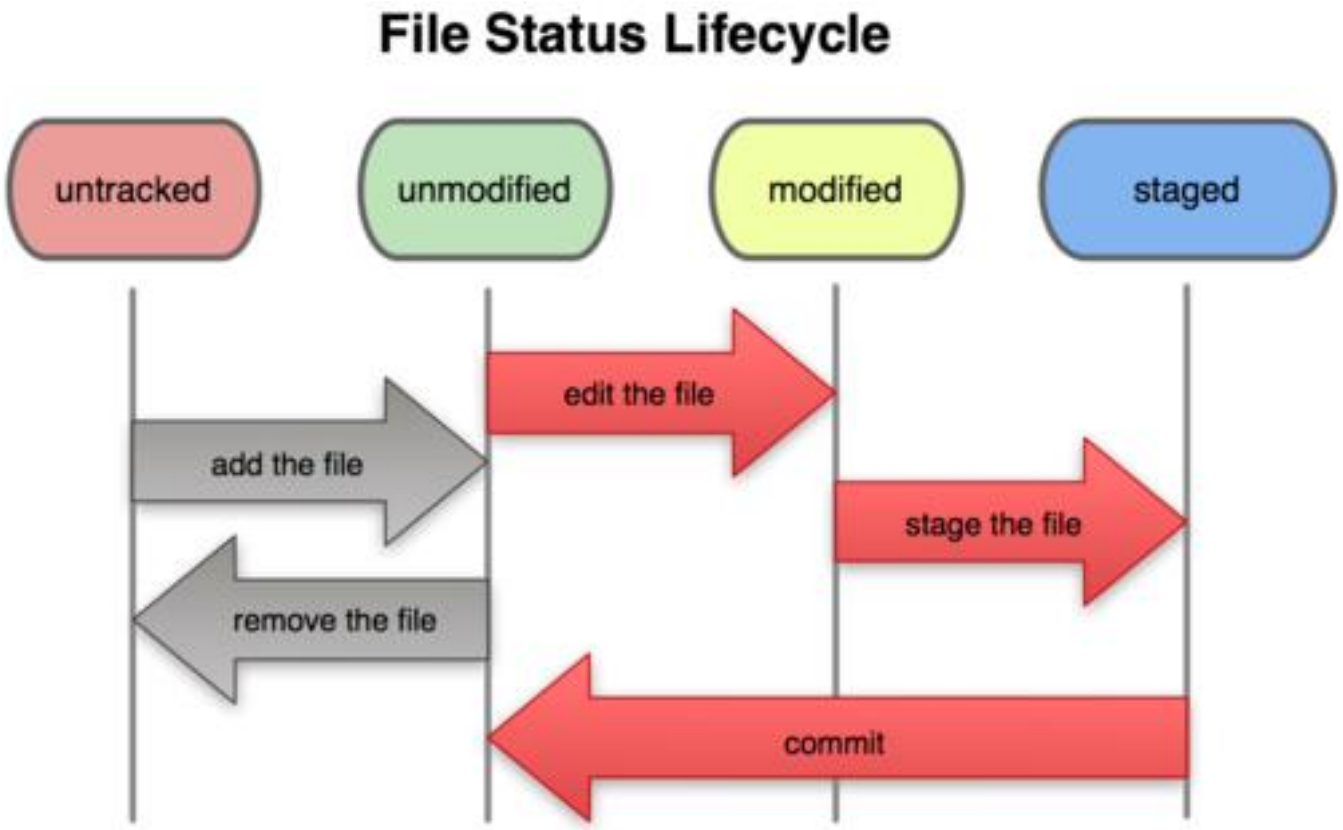
Using Git on multiple computers, multiple users or one user on multiple computers

A Local Git project has three areas



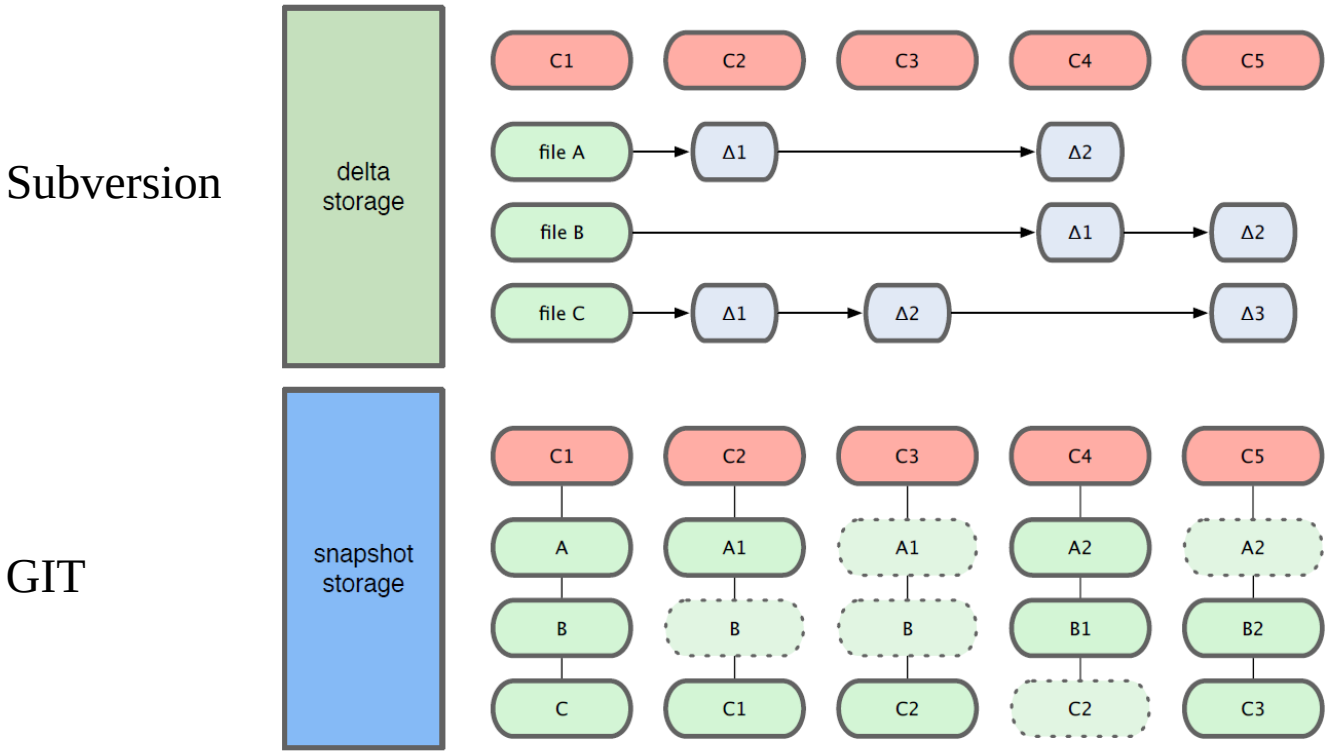
Note: working directory sometimes called the “working tree”, staging area sometimes called the “index”.

Git file lifecycle



Snapshot

- **Git use snapshot storage**



Checksum

- **Versioning**
  - **In Subversion each modification to the central repo incremented the version # of the overall repo.**
  - **How will this numbering scheme work when each user has their own copy of the repo, and commits changes to their local copy of the repo before pushing to the central server?**

- **Versioning by checksum**
  - **Git generates a unique SHA-1 hash(40 characters of hex digits) for every commit**
  - **Refer to commits by this ID rather than a version number**
  - **Example : 1677b2d, 258efa7, 0e52da7**

Basic Flow

- **A basic workflow**
  - **(Possible init or clone) Init a repo**
  - **Edit files**
  - **Stage the changes**
  - **Review your changes**
  - **Commit the changes**

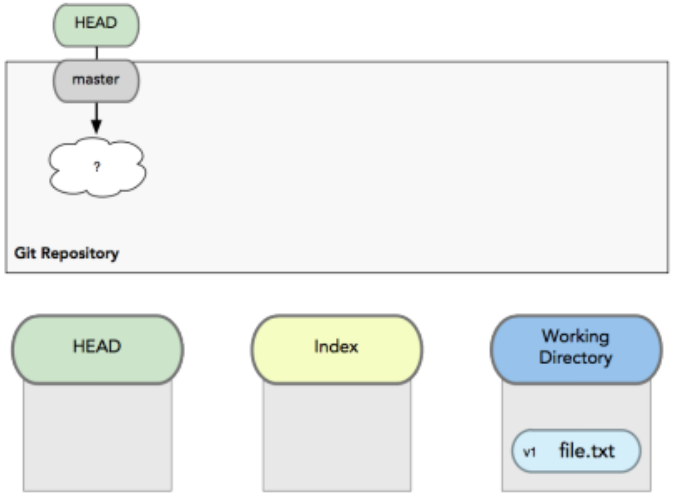
Init

- **Init a repository**
  - **git init**

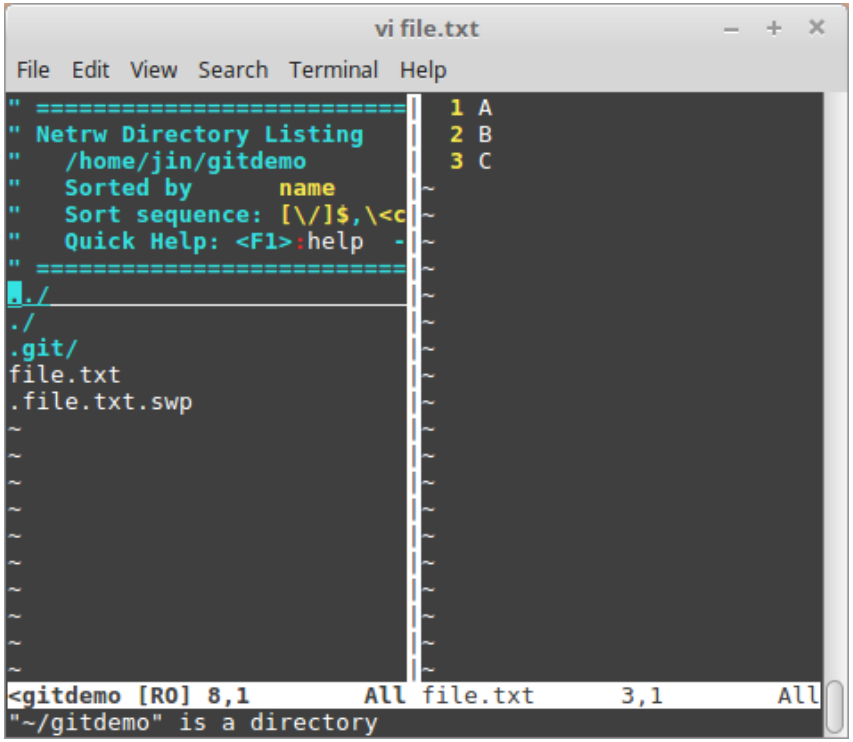
```
jin@DexterDeskTop:~/gitdemo|⇒  git init
Initialized empty Git repository in /home/jin/gitdemo/.git/
jin@DexterDeskTop:~/gitdemo|master ⇒  ls -l .git
total 32
drwxr-xr-x 2 jin jin 4096 Mar 19 12:12 branches
-rw-r--r-- 1 jin jin   92 Mar 19 12:12 config
-rw-r--r-- 1 jin jin   73 Mar 19 12:12 description
-rw-r--r-- 1 jin jin   23 Mar 19 12:12 HEAD
drwxr-xr-x 2 jin jin 4096 Mar 19 12:12 hooks
drwxr-xr-x 2 jin jin 4096 Mar 19 12:12 info
drwxr-xr-x 4 jin jin 4096 Mar 19 12:12 objects
drwxr-xr-x 4 jin jin 4096 Mar 19 12:12 refs
jin@DexterDeskTop:~/gitdemo|master ⇒
```

Edit Files

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes



- Editing

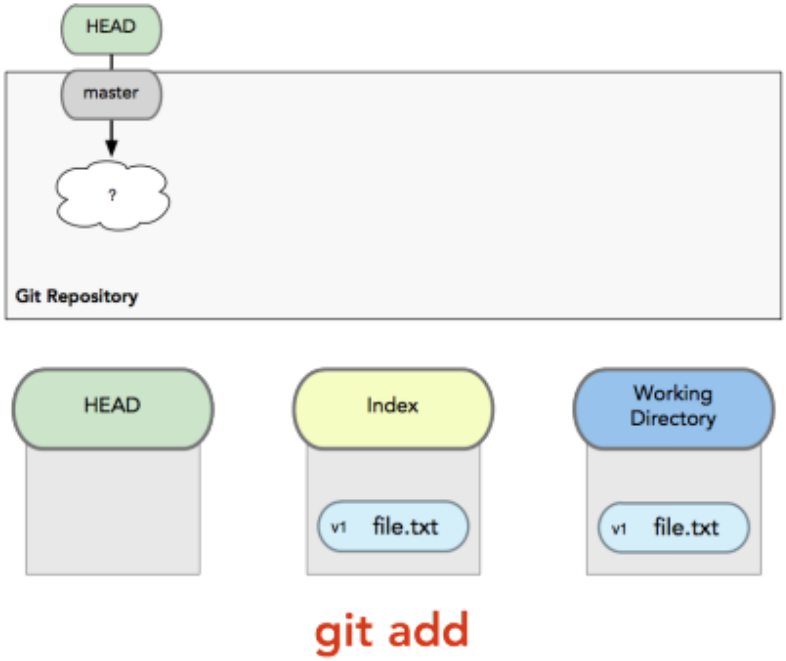


Stage the Changes

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes

```
jin@DexterDeskTop:~/gitdemo|master⚡
⇒  git add file.txt
jin@DexterDeskTop:~/gitdemo|master⚡
```

- git add filename

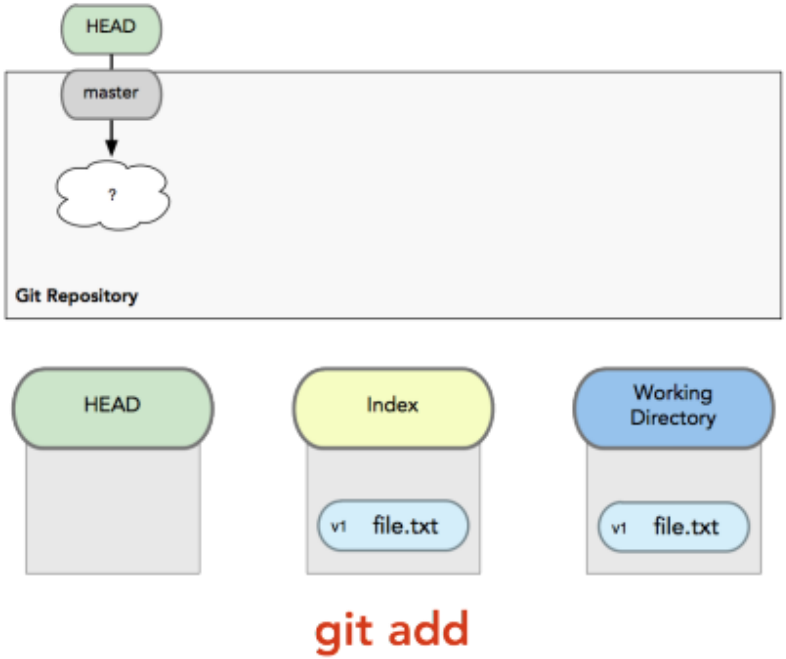


Review Your Changes

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes

```
jin@DexterDeskTop:~/gitdemo|master⚡
⇒  git add file.txt
jin@DexterDeskTop:~/gitdemo|master⚡
⇒  git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   file.txt

jin@DexterDeskTop:~/gitdemo|master⚡
```

- git status

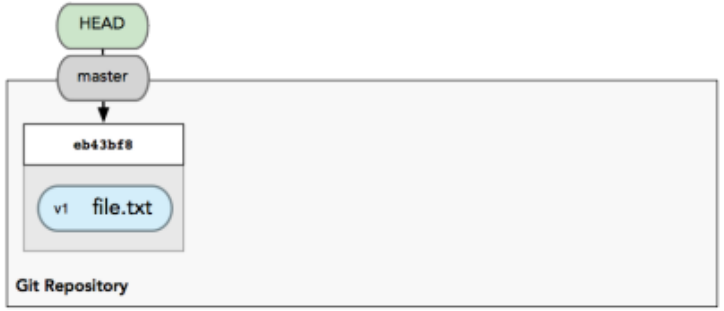


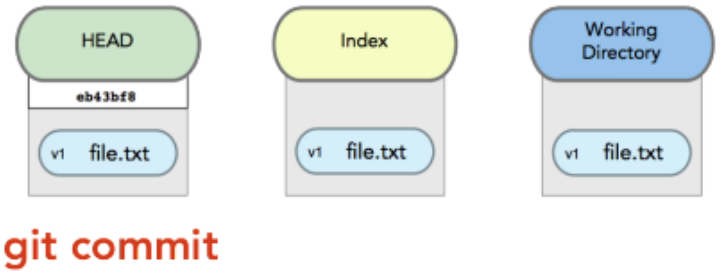
Commit the Changes

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes

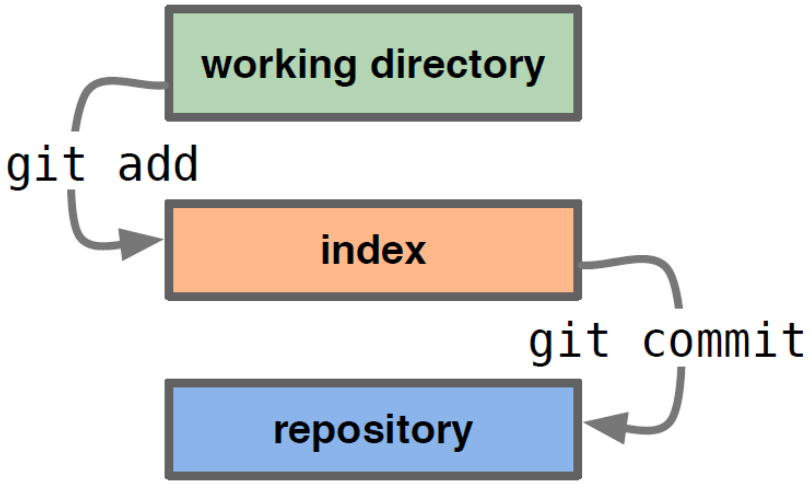
```
# Please enter the commit message for your changes. Line$
# with '#' will be ignored, and an empty message aborts $
# On branch master
#
# Initial commit
#
# Changes to be committed:
#       new file:   file.txt
#
```

- git commit





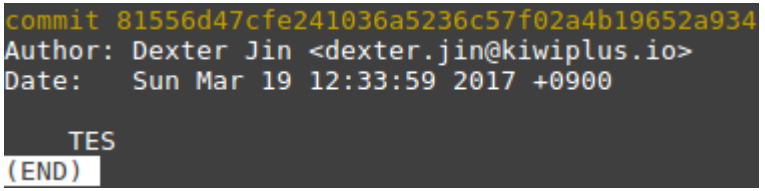
Commit the Changes



View Changes

- View changes
- git diff
  - Show the difference between working directory and staged
- git diff –cached
  - Show the difference between staged and the HEAD

- View history
- git log



Use Cases

Adding a file

- **Step**
  - **Make a file**
  - **git add <filename>**
  - **git commit –m <message>**

```
jin@Dexter:~/test|master ⇒  touch file
jin@Dexter:~/test|master⚡ ⇒  ls
file
jin@Dexter:~/test|master⚡ ⇒  git add file
jin@Dexter:~/test|master⚡ ⇒  git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   file

jin@Dexter:~/test|master⚡ ⇒  git commit -m "file added"
[master (root-commit) f9c24ca] file added
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file
jin@Dexter:~/test|master ⇒
```

Editing a file

- **Step**
  - **Modify a file**
  - **git add <filename>**
  - **git commit –m <message>**

```
nothing to commit, working directory clean
jin@Dexter:~/test|master ⇒  echo "aa" >> file
jin@Dexter:~/test|master⚡ ⇒  git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in worki
ng directory)

        modified:   file

no changes added to commit (use "git add" and/or "git commit -
a")
jin@Dexter:~/test|master⚡ ⇒  git add file
jin@Dexter:~/test|master⚡ ⇒  git commit -m "file modified"
[master 727146f] file modified
 1 file changed, 1 insertion(+)
jin@Dexter:~/test|master ⇒  git status
On branch master
nothing to commit, working directory clean
jin@Dexter:~/test|master ⇒
```

Deleting a file

- **Step**
  - **Delete a file**
  - **git add <filename>**
  - **git commit –m <message>**

```
jin@Dexter:~/test|master ⇒  rm file
jin@Dexter:~/test|master⚡ ⇒  git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed
)
  (use "git checkout -- <file>..." to discard changes in worki
ng directory)

        deleted:    file

no changes added to commit (use "git add" and/or "git commit -
a")
jin@Dexter:~/test|master⚡ ⇒  git add file
jin@Dexter:~/test|master⚡ ⇒  git commit -m "file deleted"
[master 1208526] file deleted
 1 file changed, 1 deletion(-)
 delete mode 100644 file
jin@Dexter:~/test|master ⇒  git status
On branch master
nothing to commit, working directory clean
jin@Dexter:~/test|master ⇒
```

Deleting a file

- **Step**
  - **git rm <filename>**
  - **git commit –m <message>**

```
jin@Dexter:~/test|master ⇒  git rm file
rm 'file'
jin@Dexter:~/test|master⚡ ⇒  git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    file

jin@Dexter:~/test|master⚡ ⇒  git commit -m "file deleted"
[master f547b38] file deleted
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 file
jin@Dexter:~/test|master ⇒
```

Deleting a file

- **Step**
  - **git rm –cached <filename>**
  - **git commit –m <message>**

```
jin@Dexter:~/test|master ⇒  ls
file
jin@Dexter:~/test|master ⇒  git rm --cached file
rm 'file'
jin@Dexter:~/test|master⚡ ⇒  ls
file
jin@Dexter:~/test|master⚡ ⇒  git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    file

Untracked files:
  (use "git add <file>..." to include in what will be committe
d)

        file

jin@Dexter:~/test|master⚡ ⇒  git commit -m "file deleted"
[master d32188c] file deleted
 1 file changed, 1 deletion(-)
 delete mode 100644 file
jin@Dexter:~/test|master⚡ ⇒  ls
file
jin@Dexter:~/test|master⚡ ⇒
```

***Not deleting a file in working directory**

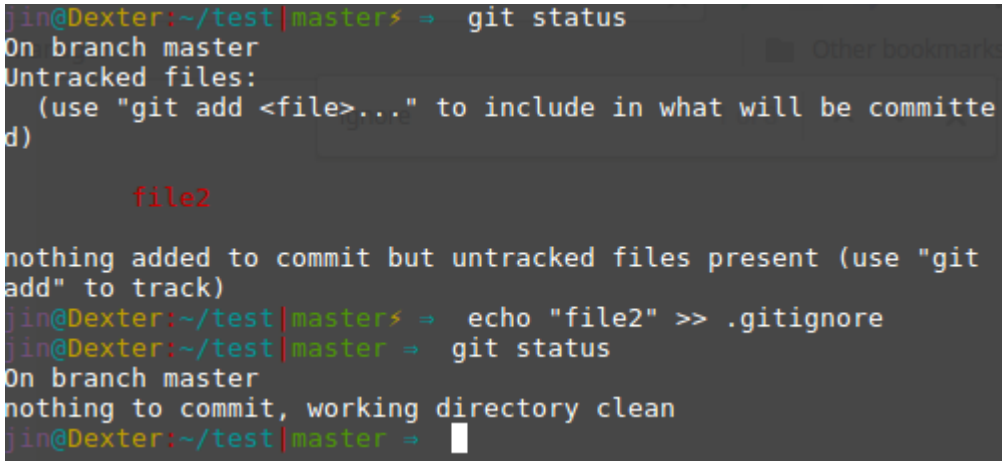
Revert to origin

- **Step**
  - **git checkout -- file**

```
jin@Dexter:~/test|master⚡ ⇒  git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in worki
ng directory)

        modified:   file

no changes added to commit (use "git add" and/or "git commit -
a")
jin@Dexter:~/test|master⚡ ⇒  git checkout -- file
jin@Dexter:~/test|master ⇒  git status
On branch master
nothing to commit, working directory clean
jin@Dexter:~/test|master ⇒
```

Ignoring a file

- **Step**
  - **Make .gitignore file**
  - **Add patterns to ignoring a file**



- **Patterns**
  - ***.o, *.exe (Ignoring build files)**