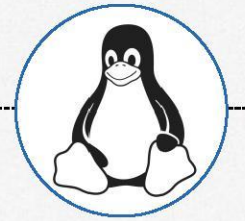
CONTENTS

HISTORY

2002

Linux kernel began using proprietary DVCS BitKeeper



2005

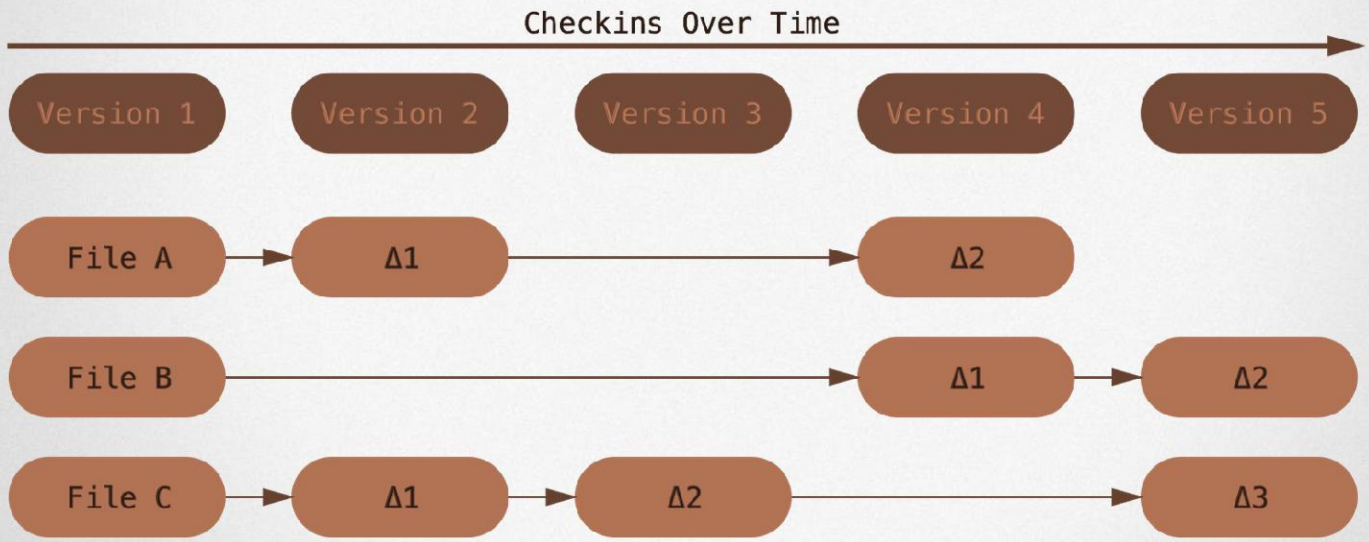
Linux kernel project stopped using BitKeeper

2005

Linus Torvalds, the creator of Linux started working on new DVCS called git

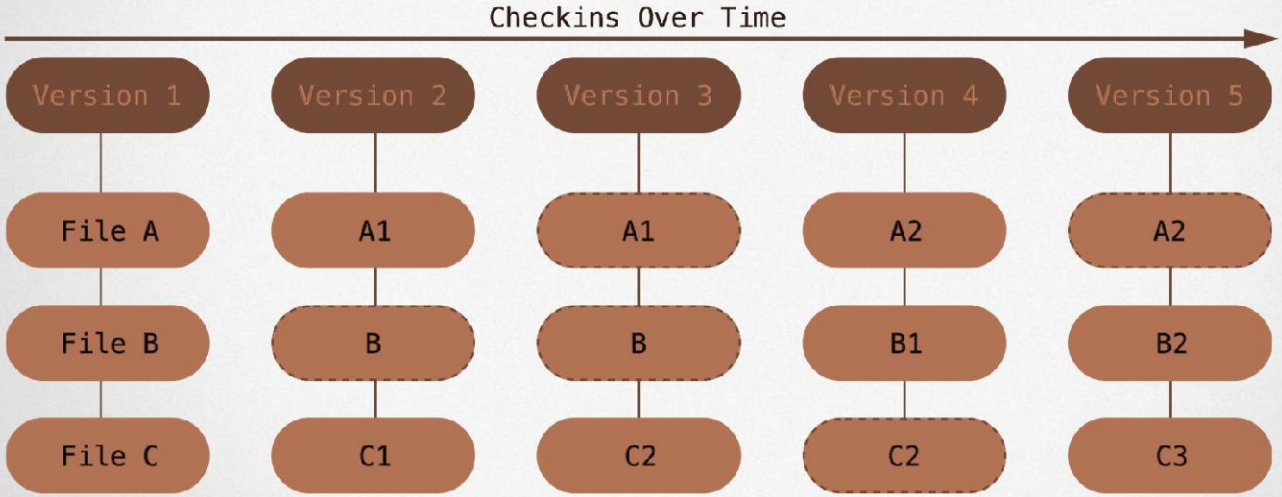
SNAPSHOTS, NOT DIFFERENCES

- Most VCS store information as a list of file-based changes. These systems think of the information they keep as a set of files and the changes made to each file over time.



SNAPSHOTS, NOT DIFFERENCES

- Git thinks of its data more like a set of snapshots of a miniature filesystem, Every time you commit in Git, it basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot



EVERY OPERATION IS LOCAL

- Most operations in Git only need local files and resources to operate – generally no information is needed from another computer on your network.
- For example, to browse the history of the project, Git doesn't need to go out to the server to get the history and display it for you – it simply reads it directly from your local database

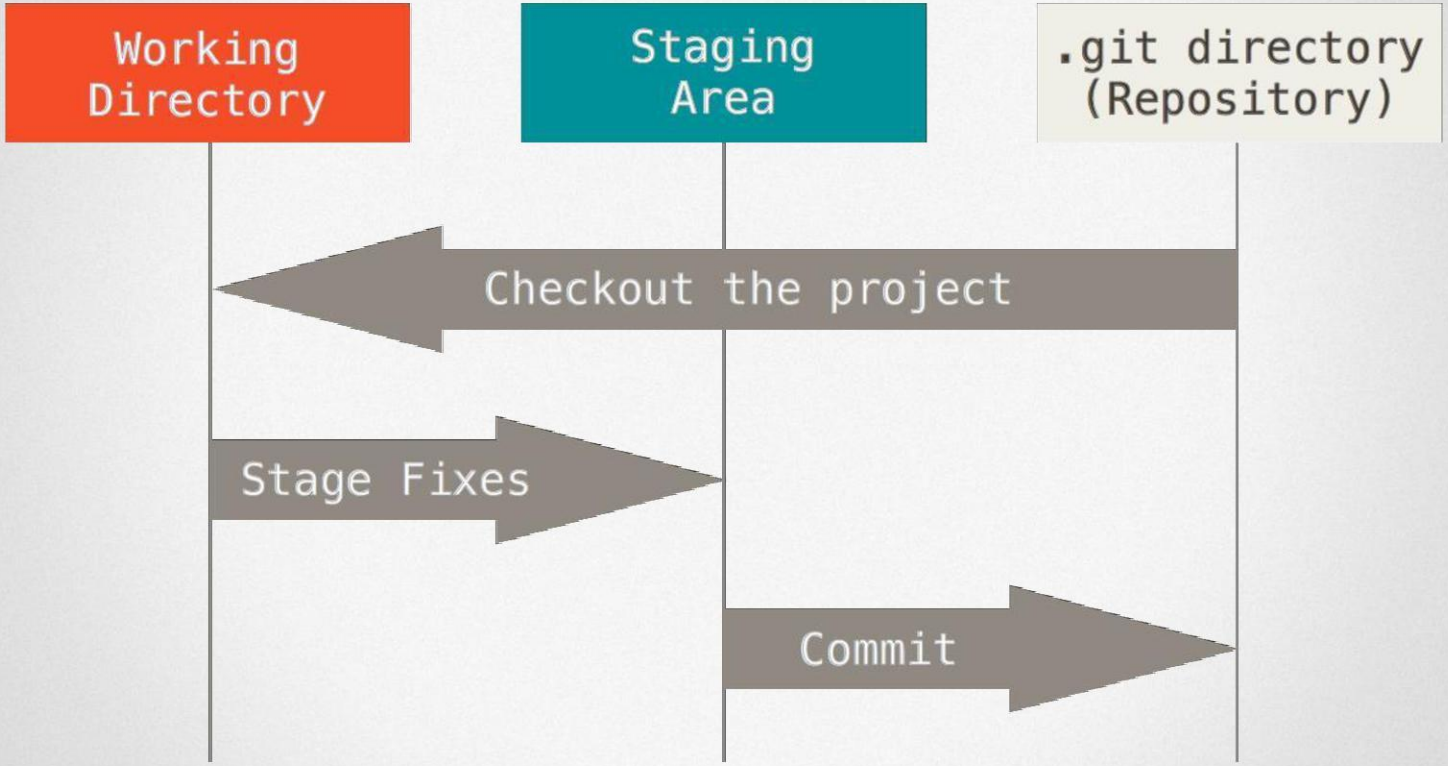
GIT HAS INTEGRITY

- Everything in Git is check-summed before it is stored and is then referred to by that checksum. This means it's impossible to change the contents of any file or directory without Git knowing about it.
- The mechanism that Git uses for this checksumming is called a SHA-1 hash. This is a 40-character string composed of hexadecimal characters (0–9 and a–f) and calculated based on the contents of a file or directory structure

THE THREE STATES

- Git has three main states that your files can reside in: **committed**, **modified**, and **staged**.
  - Committed means that the data is safely stored in your local database.
  - Modified means that you have changed the file but have not committed it to your database yet.
  - Staged means that you have marked a modified file in its current version to go into your next commit snapshot

THE THREE STATES



INSTALLING GIT

- CentOS

```
$ sudo yum install git-all
```

- Ubuntu

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

FIRST-TIME GIT SETUP

- The first thing you should do when you install Git is to set your user name and email address.
- This is important because every Git commit uses this information, and it's immutably baked into the commits you start creating

```
$ git config --global user.name "AMIR-GIU"
$ git config --global user.email "amir.haytham@giu-uni.de"
```

- You can also set your default editor and colorize the output:

```
$ git config --global core.editor vi
$ git config --global color.ui true
```

GETTING HELP

- If you ever need help while using Git, just use any of these commands:

```
$ git help <verb>
$ git <verb> --help
$ man git-<verb>
```

GETTING HELP

```
$ git help
usage: git [--version] [--exec-path[=<path>]] [--html-path]
           [-p|--paginate|--no-pager] [--no-replace-objects]
           [--bare] [--git-dir=<path>] [--work-tree=<path>]
           [-c name=value] [--help]
           <command> [<args>]
The most commonly used git commands are:
   add Add file contents to the index
   Bisect Find by binary search the change that introduced a
bug
...
```

STARTING A REPO

```
$ mkdir gittest

$ cd gittest

$ git init

Initialized empty Git repository in /home/ahaytham/gittest/.git/

$ ls -A

.git
```

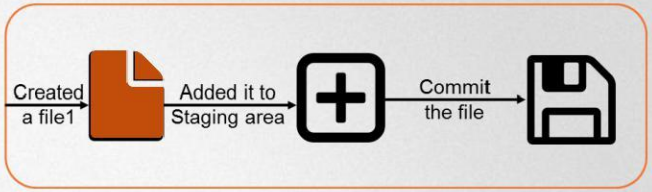
`.git` metadata directory is created!

GIT WORKFLOW

- Let's assume the following scenario:

CREATE FILE

```
$ touch file1
```



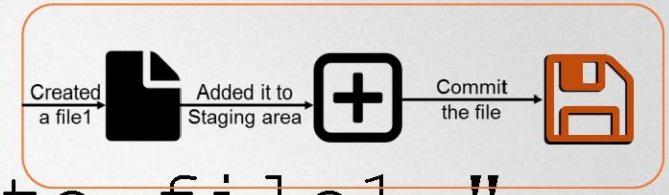

```
$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will
be committed)
#
#   file1
nothing added to commit but untracked files present
(use "git add" to track)
```

ADD TO STAGING


```
$ git add file1

$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#   new file:   file1
#
```

COMMIT CHANGES

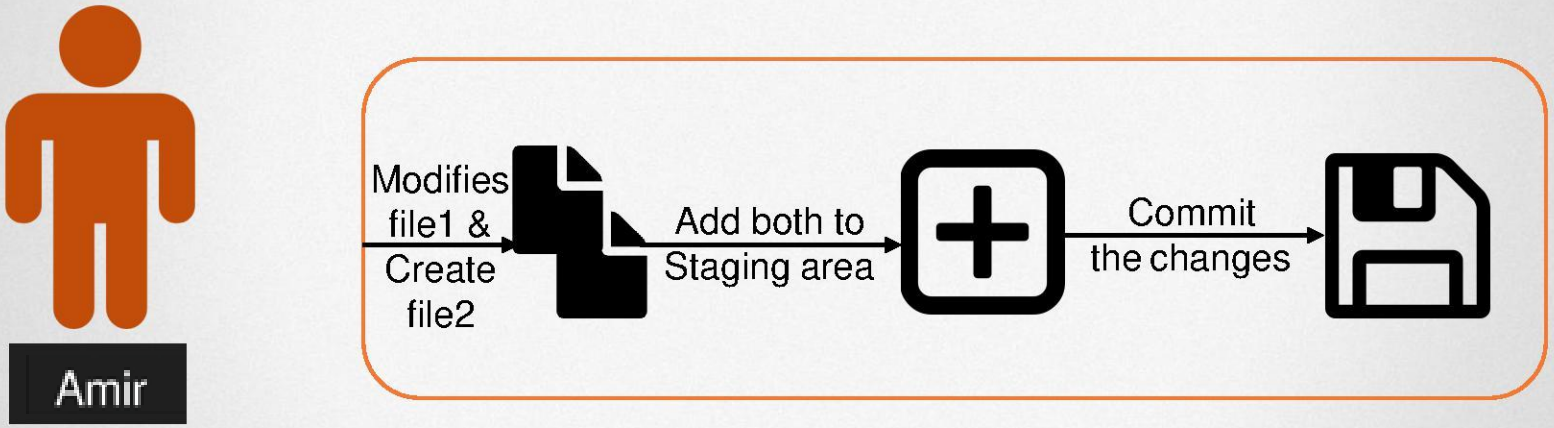



```
$ git commit -m "Create file1."
[master abe28da] Create file1.
1 files changed, 1 insertions(+), 0 deletions(-)
create mode 100644 file1
```

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

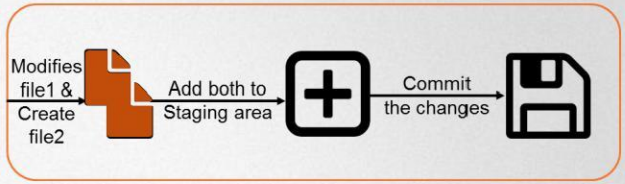
GIT WORKFLOW

- Let's assume the following scenario:



CREATE & MODIFIY FILE

```
$ touch file2
$ vi file1
$ git status
# On branch master
# Changed but not updated:
#
#   modified:    file1
#
# Untracked files:
#   (use "git add <file>..." to include in what will be
committed)
#
#   file2
nothing added to commit but untracked files present (use
"git add" to track)
```

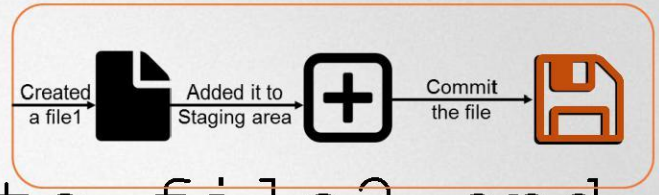


ADD TO STAGING

```
$ git add file1 file2
```


```
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#   new file:   file2
#   modified:   file1
```

COMMIT CHANGES




```
$ git commit -m "Create file2 and edit file1."
[master abe28da] Create file2 and edit file1.
2 files changed, 3 insertions(+), 0 deletions(-)
create mode 100644 file2
```

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

GIT LOG

```
$ git log
```

```
Commit 1b0019c37e3f3724fb2e9035e6bab4d7d87bf455
Author: AMIR-GIU <amir.haytham@giu-uni.de>
Date:   Thu Jul 5 22:11:27 2024 -0400

    Add file2 and edit file1.

commit 5acaf86b04aaf9cbbb8ebb9042a20a46d0b9ce76
Author: AMIR-GIU <amir.haytham@giu-uni.de>
Date:   Thu Jul 5 22:00:46 2024 -0400

    Add file1.
```

DIFFERENT WAYS TO ADD

```
$ git add <list of files>
$ git add --all
$ git add dir/
$ git add *.rb
```

GIT DIFF

- Show unstaged differences since last commit

```
$ git diff
```

- Show staged differences since last commit

```
$ git diff --staged
```

UNSTAGING FILES

- To unstage file

```
$ git reset HEAD file1
```

- Short Status

```
$ git status -s
M file1
```

DISCARD CHANGES

```
$ git status -s

M file1

$ git checkout -- file1

$ git status

nothing to commit (working directory clean)
```

SKIP STAGING

```
$ git commit -a -m "modify file1."
[master d00fefc] Modify file1.
 1 files changed, 1 insertions(+),
1 deletions(-)
```

Does not add new (untracked) file

UNDOING A COMMIT

- Move to commit before ‘HEAD’ (last commit)

```
$ git reset --soft HEAD^
```

- Undo last 2 commits

```
$ git reset --soft HEAD^^
```

- Undo the last commit and all changes

```
$ git reset --hard HEAD^
```

ADDING TO A COMMIT

- Maybe you forgot to add a file

```
$ git add file3
```

```
$ git commit --amend -m "Modify file2 & add file3."
```

ADDING/REMOVING A REMOTE

- To add remote repository

```
$ git remote add origin
https://github.com/AMIR-GIU/<repo-name>
```

- To list the remote repositories

```
$ git remote -v
```

- To delete or to rename a remote repository:

```
$ git remote rm origin
```

```
$ git remote rename origin neworigin
```

PUSHING / PULLING FROM REMOTE

- To add your files to the remote repo use this command (Note: you need to create an account on github.com)

```
$ git push origin main
```

- To get the changes made by others

```
$ git pull
```

- this command will:
  - Fetch (or Sync) our local repository with the remote one.
  - Merge the origin/master with master

PUSHING / PULLING FROM REMOTE

- Instead, you can use

```
$ git fetch
$ git merge origin/master
```

- The *fetch* command downloads the updates to a local branch called origin/master, then you have to merge the branch manually using *merge* command.

DEALING WITH CONFLICTS

```
$ git pull
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 1), reused 3 (delta 1)
Unpacking objects: 100% (3/3), done.
From https://github.com/AMIR-GIU/<project-name>
ee47baa..4e76d35  master     -> origin/master
Auto-merging file1
CONFLICT (content): Merge conflict in file1
Automatic merge failed; fix conflicts and then commit
the result
```

DEALING WITH CONFLICTS

- We have to edit the file and fix the conflict

```
<<<<<<< HEAD
This my line.
=======
No, it's mine!
>>>>>>>
4e76d3542a7eee02ec516a47600002a90a4e4b48
```

- Then commit to save the merge, then the editor will pop up to enter your commit message

```
$ git commit -a
```

CLONING A REPOSITORY

- To clone a repo

```
$ git clone https://github.com/AMIR-GIU/<repo-name>
```

- This will:
  - Download the entire repository into a new gitSandbox directory.
  - Add the ‘origin’ remote, pointing it to the clone URL

BRANCHING OUT

- If you need to work on a feature that will take some time, it's preferred to make a branch

```
$ git branch newbranch
```

- To list all branches:

```
$ git branch newbranch
* master
  newbranch
```

- To switch to a branch:

```
$ git checkout newbranch
```

CREATING A REMOTE BRANCH

- When you need other people to work on your branch.

OR

- Any branch that will last more than a day

- Then you have to make you branch available remotely:

```
$ git checkout -b newbranch
$ git push origin newbranch
```

- To list the remote branches

```
$ git branch -r
```

REMOTE SHOW

```
$ git remote show
* remote origin
Fetch URL:    https://github.com/AMIR-GIU/helloWorld.git
Push  URL:    https://github.com/AMIR-GIU/helloWorld.git
HEAD branch: master
 Remote branches:
    master              tracked
    newbranch           tracked
Local branches configured for 'git pull':
    master         merges with remote master
    newbranch      merges with remote newbranch
Local refs configured for 'git push':
master        pushes to master                  (up to date)
new branch    pushes to newbranch      (local out of date)
```

REMOVING A REMOTE BRANCH

- To delete a remote branch

```
$ git push origin :newbranch

$ git branch -d newbranch

$ git remote show origin

Remote branches:

master    tracked
refs/remotes/origin/newbranch stale (use 'git
remote prune' to remove)

$ git remote prune origin
```

MERGING BRANCHES

- After finishing your work on the branch you have to merge it with *master* branch (assuming no changes were made to the master)

```
$ git checkout master

$ git merge newbranch

Updating 1191ceb..ab48a3f Fast-
forward newfile |    1 + 1 file
changed, 1 insertion(+) create
mode 100644 newfile
```

MERGING BRANCHES

- If there were changes made to the master the merge will not run in *fast forward* mode instead it will run in *reclusive* mode and the editor will pop up to enter the merge message

```
$ git checkout master

$ git merge newbranch

Merge made by the 'recursive' strategy

…
```

CREATE / DELETE BRANCH

- When you're done with a branch, you can safely remove it

```
$ git branch -d newbranch
```

- If there were files that are not merged

```
$ git branch -D newbrnach
```

- To create a branch and checkout it in one step

```
$ git checkout -b newbranch
```

TAGGING

- A tag is a reference to a commit (used mostly for release versioning)
- There are two types of tagging
  - A **lightweight** tag is very much like a branch that doesn't change – it's just a pointer to a specific commit.
  - **Annotated** tags, however, are stored as full objects in the Git database. They're checksummed; contain the tagger name, email, and date; have a tagging message .

TAGGING

- To create a **lightweight** tag (This is basically the commit checksum stored in a file – no other information is kept).

```
$ git tag v1.4-lw
```

- To switch to a tag

```
$ git checkout v1.4-lw
```

TAGGING

- To add a **annotated** tag

```
$ git tag -a v0.0.3 -m "version 0.0.3"
```

- To push tags

```
$ git push --tags
```

REBASING

- During merge, git adds merge commits after merge this make the history of the repository ugly so we use rebase to get around that.
- So instead of running `git pull` that fetch and merge:

```
$ git fetch
$ git rebase
```

  - Move all changes to master which are not in origin/master to a temporary area.
  - Run all origin/master commits
  - Run all commits in the temporary area, one at a time

LOCAL BRANCH REBASE

- to get rid of merge commit while merging a local branch, we will do the following

```
$ git checkout newbranch
$ git rebase master
$ git checkout newbranch
$ git merge newbranch
```

HISTORY & CONFIGURATION

- To show the commits history

```
$ git log
```

- To show the history in one line

```
$ git log --pretty=oneline
```

- To format the history log

```
$  git log --pretty=format:"%h %ad- %s [%an]"
```

placeholder	replaced with
%ad	Author date
%an	author name
%h	SHA hash
%s	subject

HISTORY WITCH CHANGES

- To show the log with changes

```
$ git log --oneline -p
```

- To show the log with stats only

```
$ git log --oneline --stat
```

- To See visual representation of the branch merging into master

```
$ git log --oneline --graph
```

DATES RANGES

```
$ git log --until=1.minute.ago
$ git log --since=1.day.ago
$ git log --since=1.hour.ago
$ git log --since=1.month.ago --until=2.weeks.ago
```

DIFF

- Diff between last commit & current state

```
$ git diff HEAD
```

- Diff between current state & parent of last commit

```
$ git diff HEAD^
```

- Diff between current state & 5 commits ago

```
$ git diff HEAD~5
$ git diff f4df4rt..3df34f4
$ git diff master newbranch
```

BLAME

- Display the changes and committer name with each line of the file

```
$ git blame file1--date short
```

IGNORING FILES

- Often, you'll have a class of files that you don't want Git to automatically add or even show you as being untracked.
- In such cases, you can create a file listing patterns to match them named `.gitignore`

```
$ cat .gitignore
cache/
logs/*.log
```

IGNORING FILES

- Often, you'll have a class of files that you don't want Git to automatically add or even show you as being untracked.
- In such cases, you can create a file listing patterns to match them named `.gitignore`
- `$ cat .gitignore`

```
cache/
logs/*.log
```

UNTRACKING / DELETING FILE

- To stop tracking a file

```
$ git rm --cached file3
```

- To delete a file

```
$ git rm file2
```

CONFIG

```
$ git config --global user.name
“AMIR-GIU”

$ git config --global user.email
“”

$ git config --global core.editor
vi

$ git config --list

$ git config user.email
```

ALIASES

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