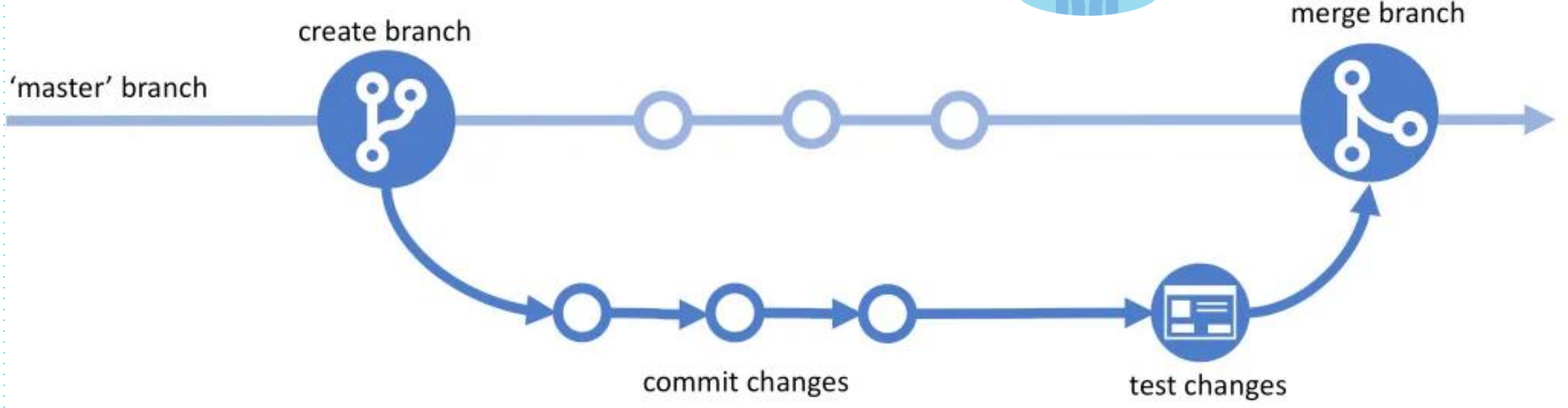
APSSDC

Andhra Pradesh State Skill Development Corporation

Skill AP
A P S S D C

VERSION CONTROL SYSTEM

Git And GitHub



Bash Commands

- **ls :** lists the contents of a directory
  - l : long directory listing
  - a : lists all files, including files which are normally hidden
  - F : distinguishes between directories and regular files
- **pwd** : prints the current working directory
- **cd** : changes directories
  - The difference between relative and absolute paths.
  - Special characters **.**, **..**, and **~**.
- **mkdir** : creates a directory
- **rmdir** : removes a directory (assuming it is empty)
  - If you get an error that the directory isn't empty even though it looks empty, check for hidden files.

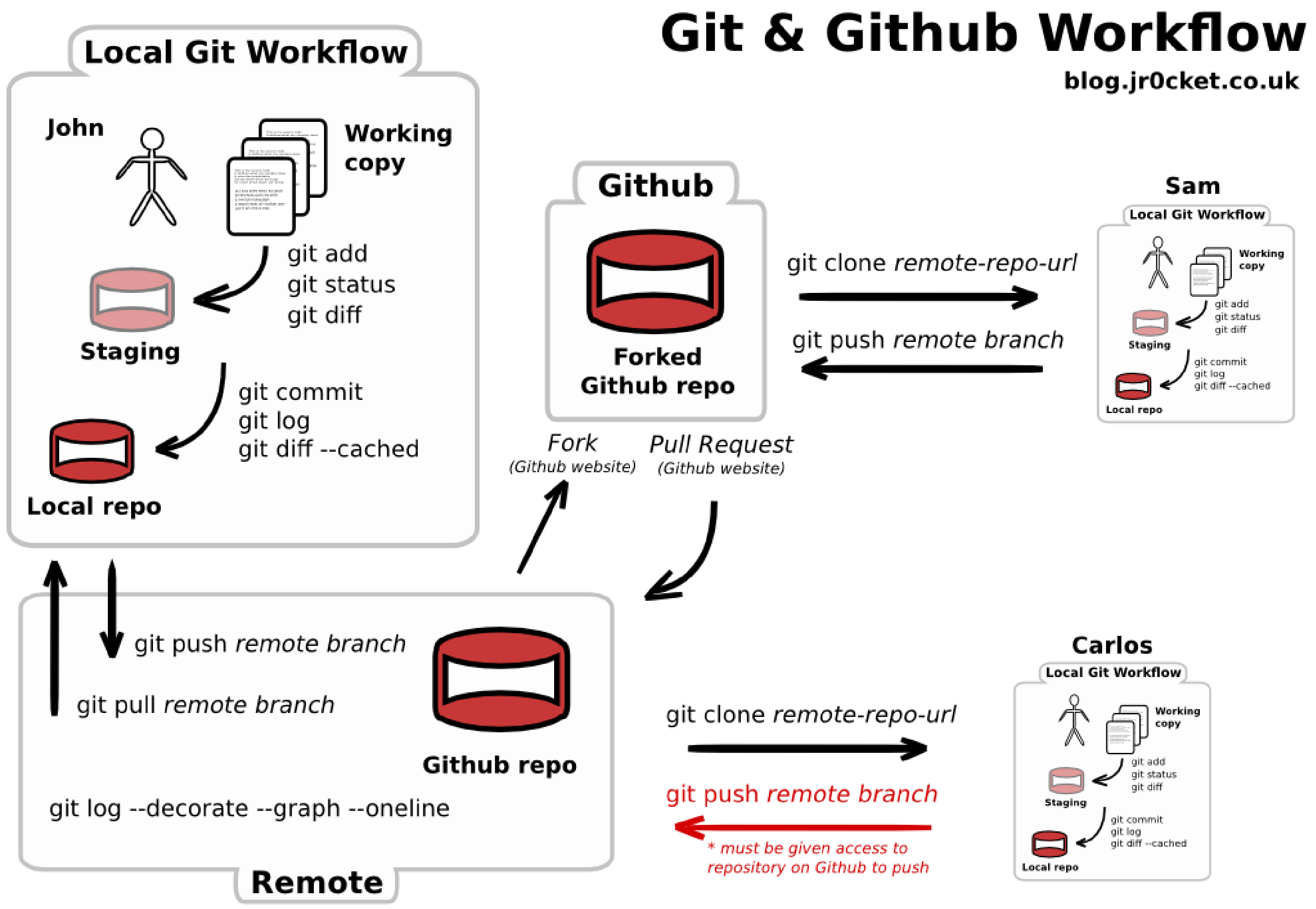
Commands

- **touch** : creates an empty file with the specified name, or if the file already exists it modifies the timestamp.
- **rm** : removes a file.
  - f : force deletion
  - r : recursive deletion
- **mv** - moves a file, or renames a file
  - f : forces overwrite, if the destination file exists
- **cp** - copies a file, leaving the original intact
  - f : forces overwrite, if the destination file exists
  - r : recursive copying of directories

Commands

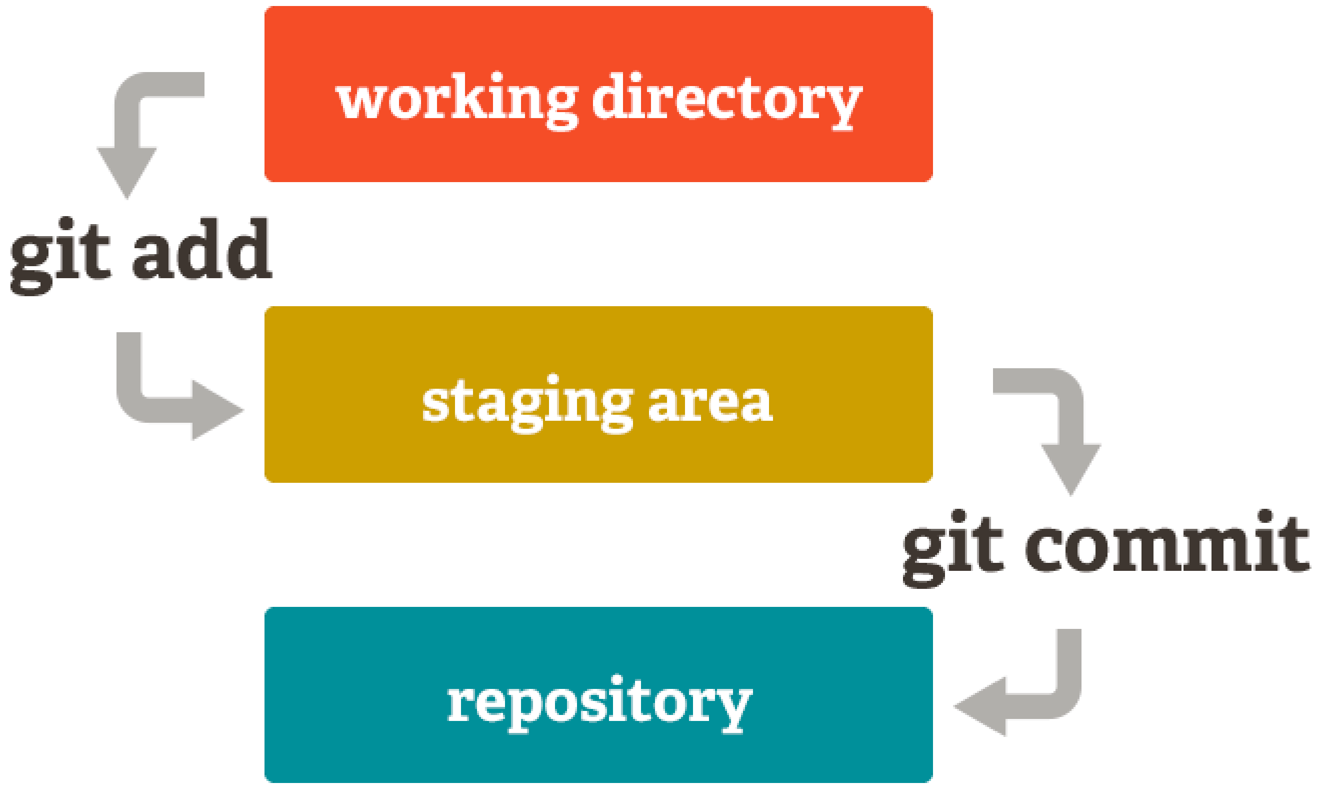
- **cat** : shows the contents of a file, all at once
- **head** : used to show so many lines form the top of a file
- **tail** : used to show so many lines form the bottom of a file
- **date** : shows the date and time on the current system
- **hostname** : prints the hostname of the current computer
- **whoami** : prints your current username

GitHub Workflow



By Anil Kumar APSSDC

Basic Git model locally



Introduce yourself to Git

- Enter these lines (with appropriate changes):
  - `git config user.name “AnilKumarTeegala"`
  - `git config user.email “anilkumar_t@apssdc.in”`
- You only need to do this once
- If you want to use a name/email address for all project,
  - `cd` to the project directory
  - Include `--global` to commands

Version Control Terminology

1. Version Control System (VCS) or (SCM)
2. Repository
3. Commit
4. SHA
5. Working Directory
6. Checkout
7. Staging Area/Index
8. Branch

Version Control Terminology

1. Version Control System :

A VCS allows you to: revert files back to a previous state, revert the entire project back to a previous state, review changes made over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more.

2. Repository:

A directory that contains your project work which are used to communicate with Git. Repositories can exist either locally on your computer or as a remote copy on another computer.

Version Control Terminology

3. Commit

Git thinks of its data like a set of snapshots of a mini file system.

Think of it as a save point during a video game.

4. SHA

A SHA is basically an ID number for each commit.

Ex. E2adf8ae3e2e4ed40add75cc44cf9d0a869afeb6

5. Working Directory

The files that you see in your computer's file system. When you open your project files up on a code editor, you're working with files in the Working Directory.

Version Control Terminology

6. Checkout

When content in the repository has been copied to the Working Directory. It is possible to checkout many things from a repository; a file, a commit, a branch, etc.

7. Staging Area

You can think of the staging area as a prep table where Git will take the next commit. Files on the Staging Index are poised to be added to the repository.

8. Branch

A branch is when a new line of development is created that diverges from the main line of development. This alternative line of development can continue without altering the main line.

Create and fill a repository

1. `cd` to the project directory you want to use
2. Type in `git init`
   - This creates the repository (a directory named `.git`)
   - You seldom (if ever) need to look inside this directory
3. Type in `git add .`
   - The period at the end is part of this command!
     - Period means “this directory”
   - This adds all your current files to the repository
4. Type in `git commit -m "Initial commit"`
   - You can use a different commit message, if you like

Clone a repository from elsewhere

- `git clone` *`URL`*
- `git clone` *`URL mypath`*
  - These make an exact copy of the repository at the given URL
- `git clone git://github.com/`*`rest_of_path/file.git`*
  - GitHub is the most popular (free) public repository
- All repositories are equal
  - But you can treat some particular repository (such as one on GitHub) as the "main" directory
- Typically, each team member works in his/her own repository, and "merges" with other repositories as appropriate

The repository

- Your top-level **working directory** contains everything about your project
  - The working directory probably contains many subdirectories—source code, binaries, documentation, data files, etc.
  - One of these subdirectories, named `.git`, is your repository
- At any time, you can take a "snapshot" of everything (or selected things) in your project directory, and put it in your repository
  - This "snapshot" is called a commit object
  - The commit object contains (1) a set of files, (2) references to the "parents" of the commit object, and (3) a unique "SHA1" name
  - Commit objects do *not* require huge amounts of memory
- You can work as much as you like in your working directory, but the repository isn't updated until you `commit` something

init and the .git repository

- When you said `git init` in your project directory, or when you cloned an existing project, you created a repository
  - The repository is a subdirectory named .git containing various files
  - The dot indicates a “hidden” directory
  - You do *not* work directly with the contents of that directory; various git commands do that for you
  - You *do* need a basic understanding of what is in the repository

Making commits

- You do your work in your project directory, as usual
- If you create new files and/or folders, they are *not tracked* by Git unless you ask it to do so
  - `git add` *`newFile1 newFolder1 newFolder2 newFile2`*
- Committing makes a "snapshot" of everything being tracked into your repository
  - A message telling what you have done is required
  - `git commit –m "Uncrevulated the conundrum bar"`
  - `git commit`
    - This version opens an editor for you the enter the message
    - To finish, save and quit the editor
- Format of the commit message
  - One line containing the complete summary
  - If more than one line, the second line must be blank

Working with your own repository

- A head is a reference to a commit object
- The "current head" is called HEAD (all caps)
- Usually, you will take HEAD (the current commit object), make some changes to it, and commit the changes, creating a new current commit object
  - This results in a linear graph: A → B → C → …→ HEAD
- You can also take any previous commit object, make changes to it, and commit those changes
  - This creates a branch in the graph of commit objects
- You can merge any previous commit objects
  - This joins branches in the commit graph

Commit messages

- In git, “Commits are cheap.” Do them often.
- When you commit, you must provide a one-line message stating what you have done
  - Terrible message: “Fixed a bunch of things”
  - Better message: “Corrected the calculation of median scores”
- Commit messages can be very helpful, to yourself as well as to your team members
- You can’t say much in one line, so commit often

Choose an editor

- When you “commit,” git will require you to type in a commit message
- For longer commit messages, you will use an editor
- The default editor is probably `vim`
- To change the default editor:
  - `git config --global core.editor /path/to/editor`
- You may also want to turn on colors:
  - `git config --global color.ui auto`

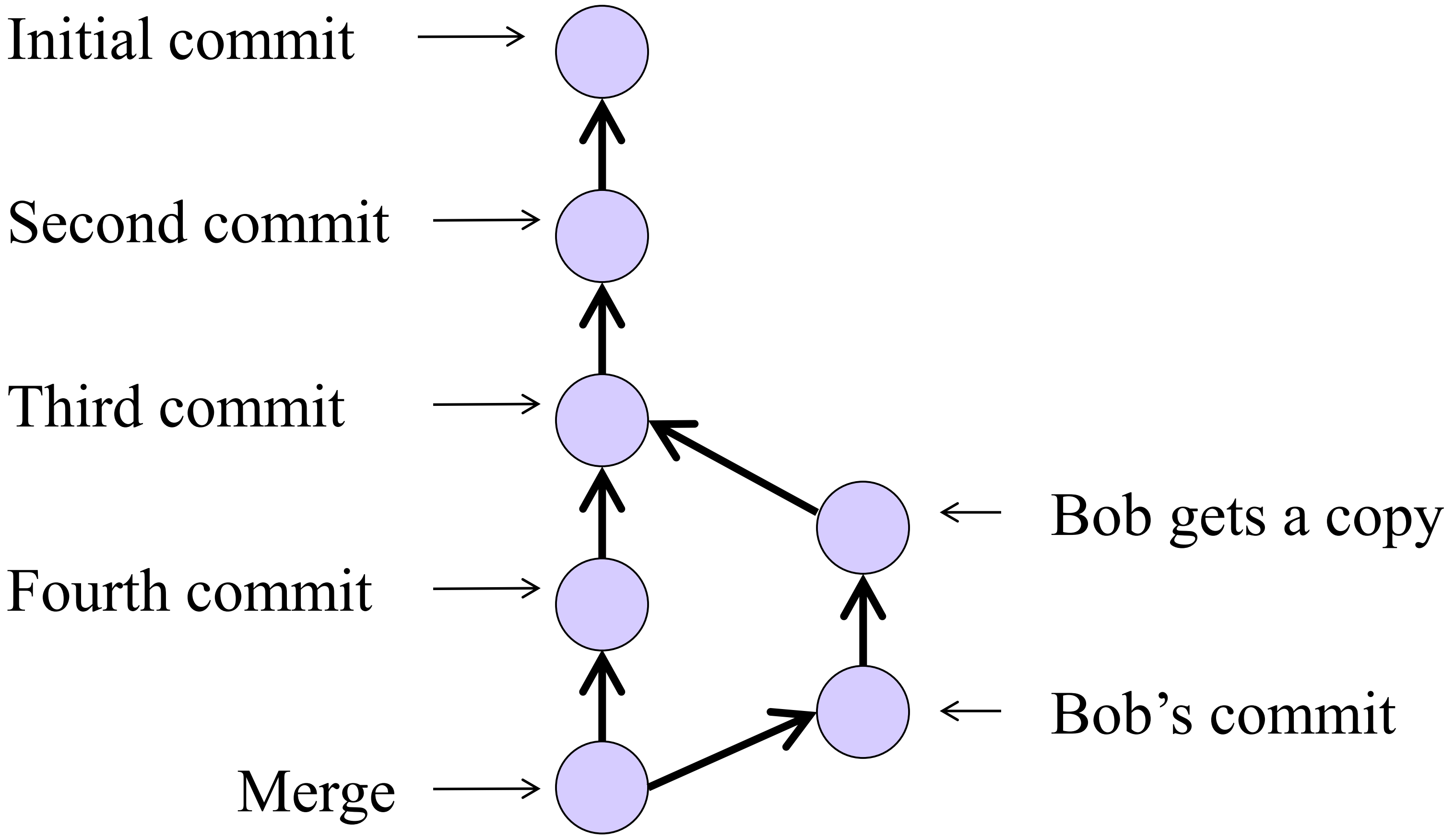
Working with others

- All repositories are equal, but it is convenient to have one central repository in the cloud
- Here’s what you normally do:
  - Download the current HEAD from the central repository
  - Make your changes
  - Commit your changes to your local repository
  - Check to make sure someone else on your team hasn’t updated the central repository since you got it
  - Upload your changes to the central repository
- If the central repository *has* changed since you got it:
  - It is *your* responsibility to **merge your two versions**
    - This is a strong incentive to commit and upload often!
  - Git can often do this for you, if there aren’t incompatible changes

Typical workflow

- `git pull` *`remote_repository`*
  - Get changes from a remote repository and merge them into your own repository
- `git status`
  - See what Git thinks is going on
  - Use this frequently!
- Work on your files (remember to `add` any new ones)
- `git commit -m` *“What I did”*
- `git push`

Multiple versions



Keeping it simple

- If you:
  - Make sure you are current with the central repository
  - Make some improvements to your code
  - Update the central repository before anyone else does
- Then you don’t have to worry about resolving conflicts or working with multiple branches
  - All the complexity in git comes from dealing with these
- Therefore:
  - Make sure you are up-to-date before starting to work
  - Commit and update the central repository frequently
- If you need help: https://help.github.com/

The End

When I say I hate CVS with a passion, I have to also say that if there are any SVN [Subversion] users in the audience, you might want to leave. Because my hatred of CVS has meant that I see Subversion as being the most pointless project ever started. The slogan of Subversion for a while was "CVS done right", or something like that, and if you start with that kind of slogan, there's nowhere you can go. There is no way to do CVS right.

--Linus Torvalds, as quoted in Wikipedia