Introduction

- Who Are We?
- What
- Where
- When
- Why
- How

Dallas Hands-On-Java

- Who Are We?
- What
- Where
- When
- Why
- How

Dallas Hands-On-Java

A group of professionals pursuing a hands-on knowledge of Java technologies.

There are hundreds of us in various disciplines and levels of knowledge in the area !!!

Dallas Hands-On-Java

- Who
- **What We Do**
- Where
- When
- Why
- How

- We provide a venu to learn-and-try-out ideas and technology
- We provide career strengthening through monthly meetings and networking collaboration with peers
- Alignment with other professions that have roles that interact with IT / Lean / Agile Software in the workplace

Dallas Hands-On-Java

- Who
- What
- Where We Bring Value
- When
- Why
- How

We bring value in teaching skills,

promoting the value of Best-of-Breed

practices, processes, tools.

=> Raising the standard and quality of our capabilities in our work place.

=> Providing an environment to bring any question, discuss ideas, be inquisitive and **try-before-you-buy** !

Dallas Hands-On-Java

- Who
- What
- Where

The Dallas Hands-On-Java meets on

the 4th Wednesday of every Month.

- When Do We Do All This?
- Why
- How

When we aren't meeting together, we are working with individuals and our organizations to foster growth in skills and knowledge through training, mentoring, professional alliances and networking

Dallas Hands-On-Java

- Who
- What
- Where
- When

- To Promote Soft Skill Development
- To Guide an Emerging Role that's Part of Every Facet of Business – Question, Learn, Apply
- To foster Consistency and Professionalism in the Dallas area

- **Why Do We Do All This?**
- How

- To Ensure that Good Developers Have Everything they Need to Succeed

Dallas Hands-On-Java

- Who
- What
- Where
- When
- Why
- How Does this all Happen?

The Dallas Hands-On-Java works as an all volunteer organization to encourage participation, interaction, involvement and commitment to each other's professional development and the empowerment of the IT / Lean / Agile Software community at large.

Dallas Hands-On-Java

- Current email address March 2012: dallashandsonjava@googlegroups.com
- Current web address: http://groups.google.com/group/dallashandsonjava
- Our GitHub **DallasHandsOnJava**
-

Dallas Hands-On-Java

Sponsor /Announcement / Thanks

Matrix Resources

5151 Beltline, Suite 1010 (10th Floor)

Addison, Tx 75252

- Thanks Matt Castleberry and Justin Thomason
- NOTE: **bathroom's being upgraded** – plan ahead to go to floor above ! (ask if elevator is restricted)

Dallas Hands-On-Java

Why are we here today?

To Learn why and how GIT aids software development.

The outcome of this tutorial:

Understand the design and philosophy of GIT
Be able to use GIT for everyday work

Our Mission

Dallas Hands-On-Java

- Who
- **What**
- Where
- When
- Why
- How

- Revision control (RCS), Software configuration management (SCM), or configuration management (CM)
- Source code management (also SCM)
- Source code control, or source control
- Version control (VCS)
- Some people claim that these terms actually have different meanings, I've found in practice, they overlap so much that there's no agreed or even useful way to tease them apart.

Revision Control System

Who

- What
- Where
- When
- Why
- How

Collaboration is ubiquitous in software development. Consequently, you see ClearCase, CVS, SVN, Mercurial, TLA, GIT and others.

How can a SCM help many people to work together effectively?

The past is a foreign country; they do things differently there.

- L. P. Hartley

- Who
- What
- Where did they all come from and Why ?
- When
- Why
- How

Why change? … Some Thoughts

- An opportunity to offer a proprietary advantage that would be hard to duplicate in competing systems
- A wish for some functionality that was not present in existing systems
- It is easier to create a new system than change existing systems – it implies fundamental architectural differences, which implies a high cost

First Generation - 1970s

- Conflict resolution using locking-based
- 'local-access' needed to a file (no client-server)
- All file-oriented
- Merging-based systems arriving towards the end of era.

SCCS - Bell Labs, in the early 1970s

- SCCS - the first version control system.
- It only versions **individual files**.
- Revision storage format not widely supported, is called "interleaved deltas." The **time to reconstruct** any revision is proportional to the total size of all revisions.
- **Branches** on the latest revision of a file are supported (called "Releases").
- **Merging is not Supported**

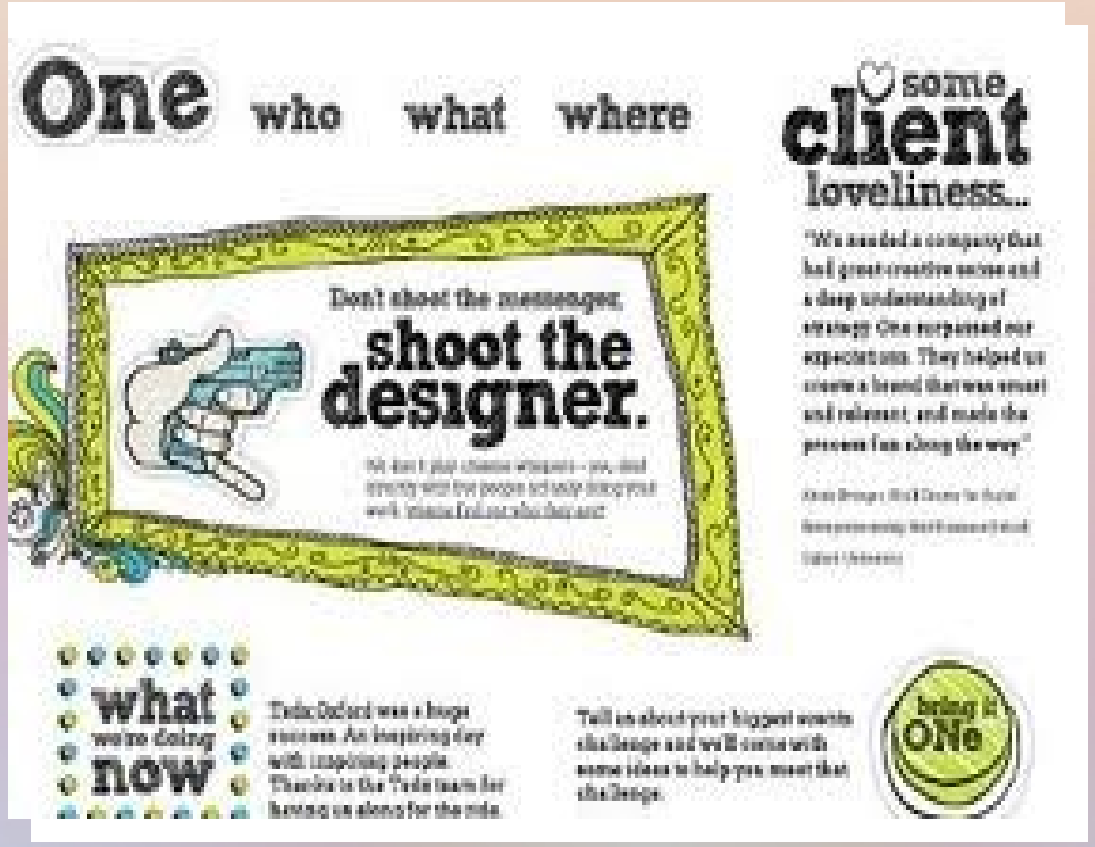
RCS - Walter Tichy, free early 1980s

- Like SCCS, Revision Control System required developers to work in a **single shared workspace**, and to lock files to prevent multiple people from modifying them simultaneously.
- RCS is more efficient than SCCS for common operations such as checking out the most recent version. W. Tichy at Purdue inverted the order of differences in the ,v file.
- Supports branches within a file; allows merges.
- **Locking is the only mechanism** for conflict prevention.
- RCS commands are configured to a particular RCS file.
- Competitors: IBM CLEAR/CASTER, AT&T SCCS (already discussed), CMU Software Development Control system and DEC Code Management System

DSEE/ClearCase

- DSEE (Domain Software Engineering Environment) was a VCS with **significant SCM features** including a **build engine** and a **task tracker**.
- First released in 1984 on Apollo Domain workstations, Apollo was acquired by HP in 1989, DSEE team spun off and became Atria; DSEE was rebranded as ClearCase. Atria acquired by IBM; continues to be in wide use; in 2008 named Rational ClearCase
- **Proprietary and closed source**

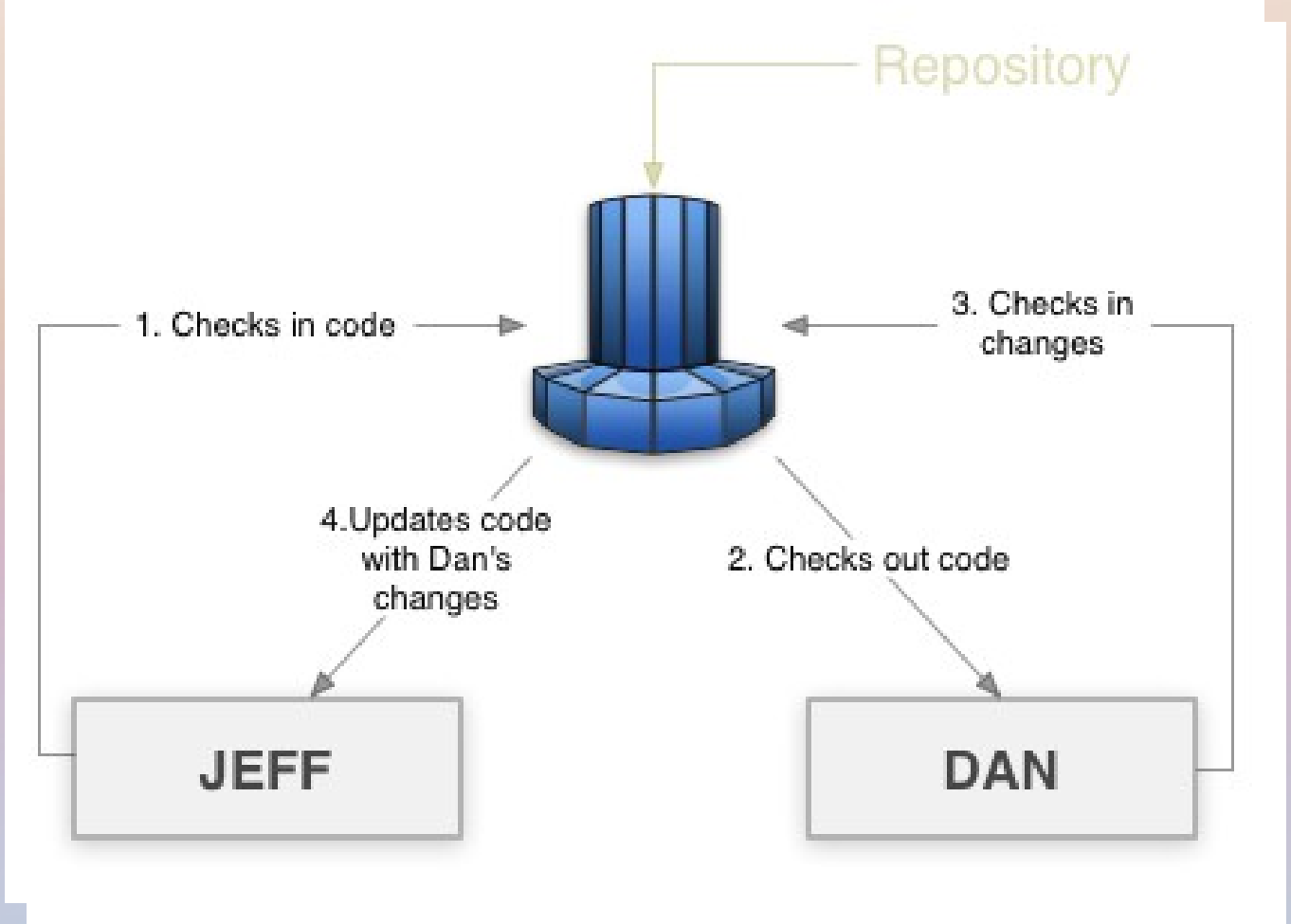
And today we would say...



Second Generation – mid 80s – 2000 +

- **Client-server** - Centralized
- Conflict Resolution: **Merge-before-commit** or Locking
- Units of operation: Some files / some **Filesets**
- 'snap-shot' of data model stored

Central 'lock' model



Concurrent Versions System (CVS) ~ 1987

- CVS was file-oriented and centralized, like its predececessors, but **broke new ground--designed for collaborative development**; using a merging rather than locking-based approach. Almost all subsequent systems have followed this lead, adopting CVS terminology and conventions and even the style of its command-line interface
- CVS **open-source** from the beginning; installed on Linux and Unix systems today.
- **Moving or renaming files** in a CVS archive was **poorly supported** and **could cause serious trouble**.
- Underlying problem was the file-oriented assumptions baked into CVS's lower layers.
- **Late 1990s** – clear for years that CVS's architecture was simply not strong enough; **developers began casting about for a successor.**

Subversion (SVN)

- In 2000, some former CVS core developers launched Subversion; **first official production release did not arrive till 2004**
- **clearly the best of the centralized** VCSes, but just as **clearly the last of its kind**.
- SVK, a layer on top of Subversion that implements decentralized/disconnected operation and more powerful merging operations, has not 'taken-off'.
- As dispersed, Internet-mediated software development becomes the norm rather than the exception, the **centralized model** of version control is **running out of steam** as visibly as the file-centric one did with CVS in the 1990s.

Are we living with this today?

Third Generation – '97 to 2003+

- Natively **Decentralized**
- Conflict Resolution: Some commit-before-merge, some merge-before-commit
- Operate on **Filesets**
- Commit **Atomically**
- **Many competing projects** in this space; BitKeeper, GNU Arch, git, monotone, darcs, Mercurial (hg), and bzr

Presenting ….

GIT

- It is open source and free
- Git, initially designed and developed in 2005 by Linus Benedict Torvalds (b. 12/28/1969)
- Designed as a way of recovering from a BitKeeper fiasco for Linux kernel development.
- Its first usage was for Linux and for Git itself.

A Fast Version Control System

- Git
  - Is **distributed**
  - Has **no master** copy, everyone is on their own island
  - Has fast merges
  - Is controversial
  - Scales up, branching and merge is cheap
  - Convenient tools still being built
  - Safeguards against corruption
  - Every commit has a unique checksum
  - Separates Author vs. Committer

Dallas Hands-On-Java

Want to Learn More – Hands-On ?

Dallas Hands-On-Java

Willing to make mistakes – Hands-On ?

Dallas Hands-On-Java

Want to Learn More – Hands-On ?

- Then 'yall better fire up them thar Laptops because here we go …. Let's get noticed at work !

Install Git

- http://help.github.com/win-set-up-git/
- http://help.github.com/linux-set-up-git/
- http://help.github.com/mac-set-up-git/

Next: Set Up SSH Keys

- Generally follow:

  1) Check for existing Keys (public/private)

  2) Backup and remove existing Keys

  3) Generate new SSH Keys (public/private)

  4) Add / submit your public SSH key to the repository you want to collaborate with

  5) Test out the access

Then: Set Up Your Info

- Set your username and email.

```
$ git config --global user.name "Firstname Lastname"
$ git config --global user.email "your_email@youremail.com"
$ git config --list
$ git help config
```

- Many, Many other git config options:

```
$ git config --global core.filemode false
$ git config --global core.autocrlf false
$ git config --global alias.ci "commit"
$ git config --global core.editor vi
$ git config --global merge.tool vimdiff
```

Set your GitHub token

- Set your GitHub token – Some tools connect to GitHub without SSH. To use these tools properly you need to find and configure your API Token.

Create a Repository (local)

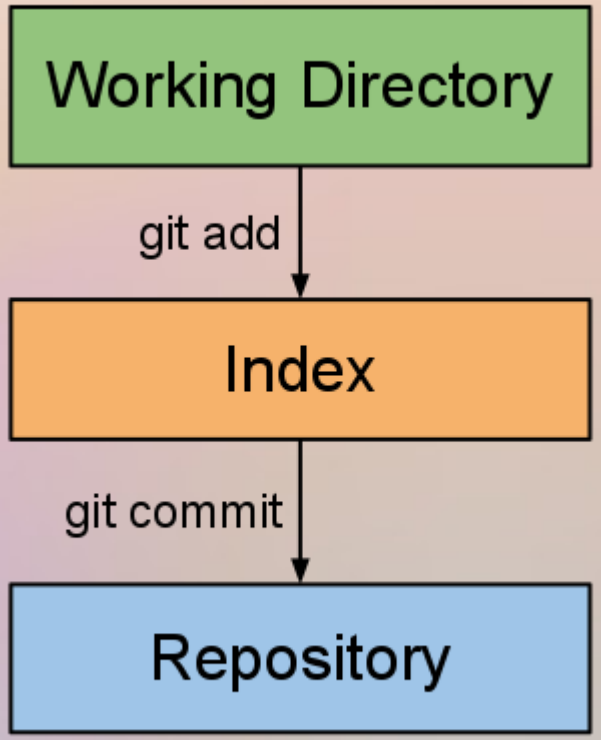
- `$ mkdir dhoj01`
  `$ cd dhoj01`
  `$ git init`
  [git status]
- `$ touch README`
  [git status]
- `$ git add README`
  [git status]
- `$ touch .gitignore`
-

Create a Repository (local) – Step 2

- Create/edit the new **.gitignore** file in the root of /path/to/my/codebase with the following contents:

```
# Ignore Eclipse configuration files
.classpath
.project
.settings/
.metadata/
# Ignore Maven target directories
target/
# Ignore common temporary files
.tmp*
```

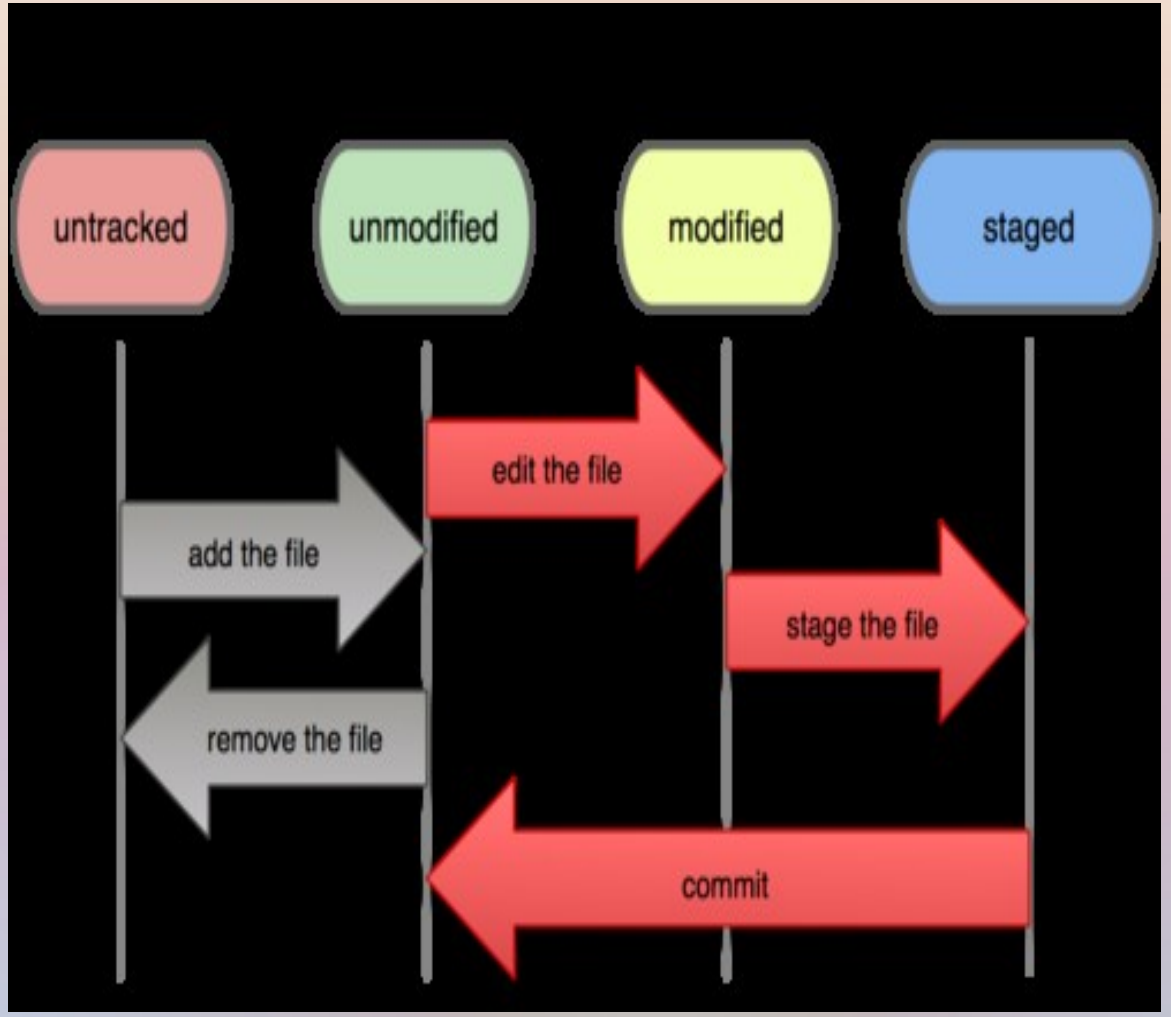
Add / Commit a Repository (local)



Create a Repository (local) – Step 3

- `$ git status`
  [look at different states]
- `$ git add .gitignore`
  [git status]
- `$ git commit -m 'first commit'`
  [git status and git branch]
-
- OPTIONAL: cat .git/config

File Status Lifecycle

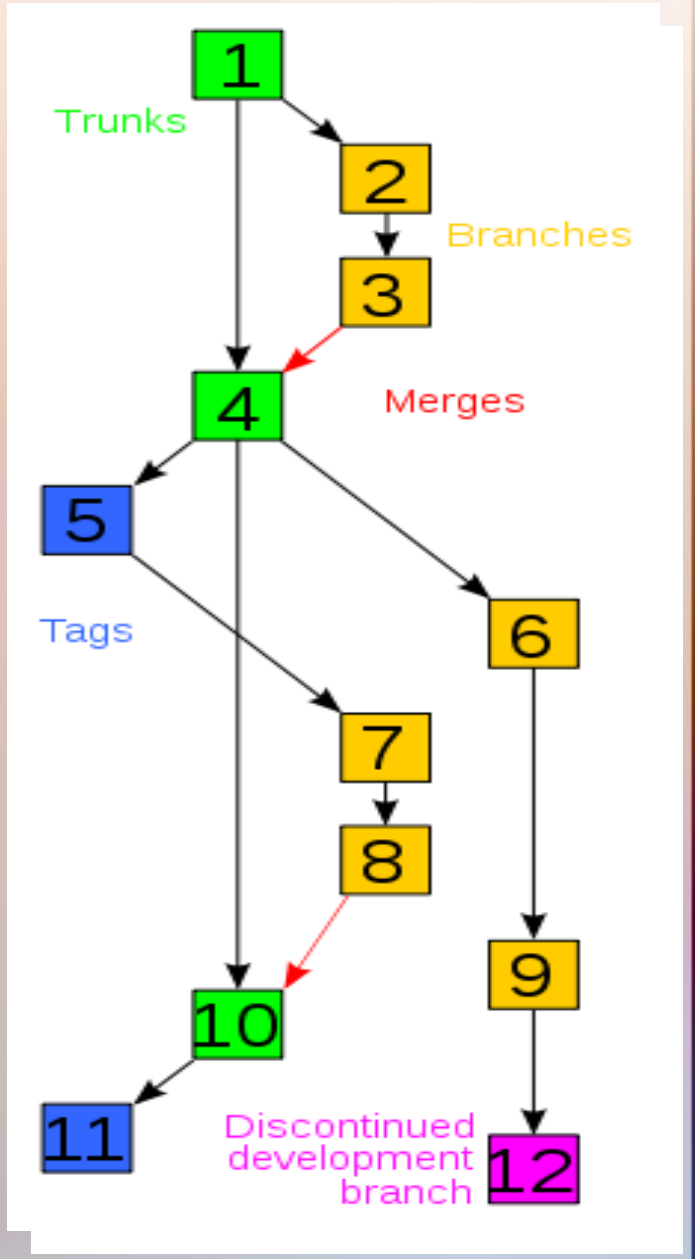


Every Day Tasks / Terms

- Head
- Branch
- Node / Hash / SHA-1
- Viewing Logs / History

Head: you see that branch is a series of commits. A branch name also means the latest commit on that branch

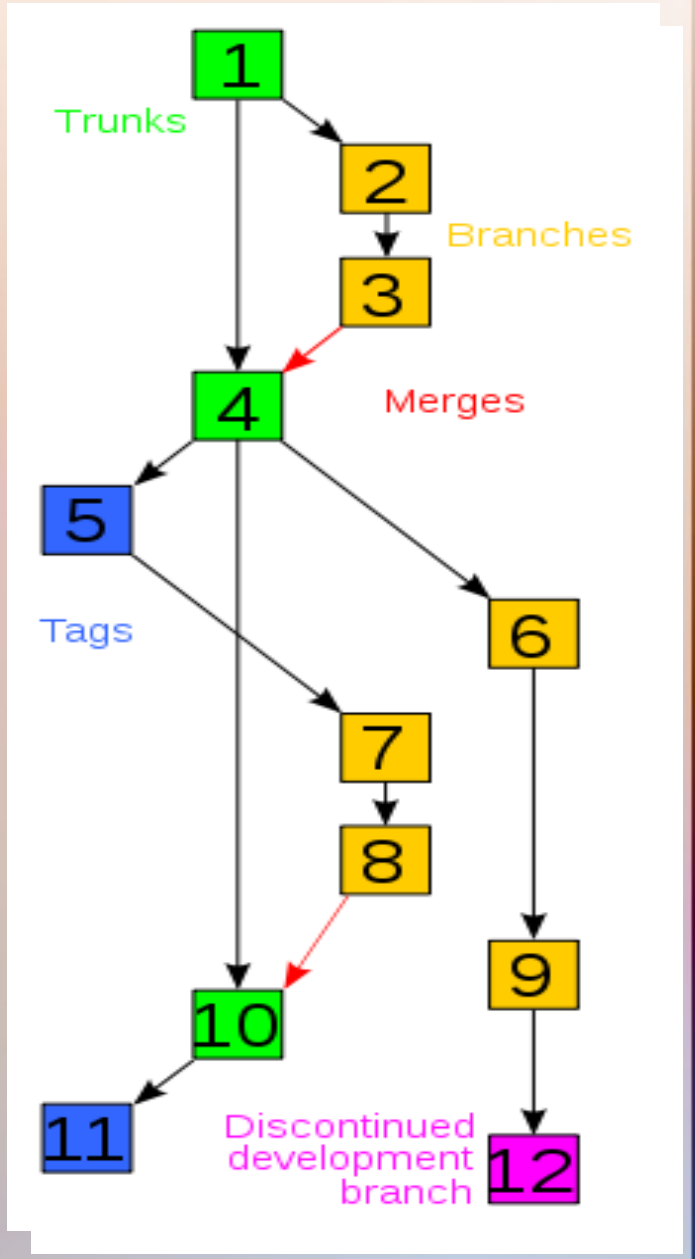
- HEAD: the commit that you are working on
- HEAD^, HEAD^^, HEAD~3 = HEAD^^^,
- HEAD^1, HEAD^2



Branches allow you to track changes with different histories.

In Git everything is treated like a branch. All the work is by default done in the master branch, but nothing is stopping your from giving it another name.

By default the main 'trunk' is called master

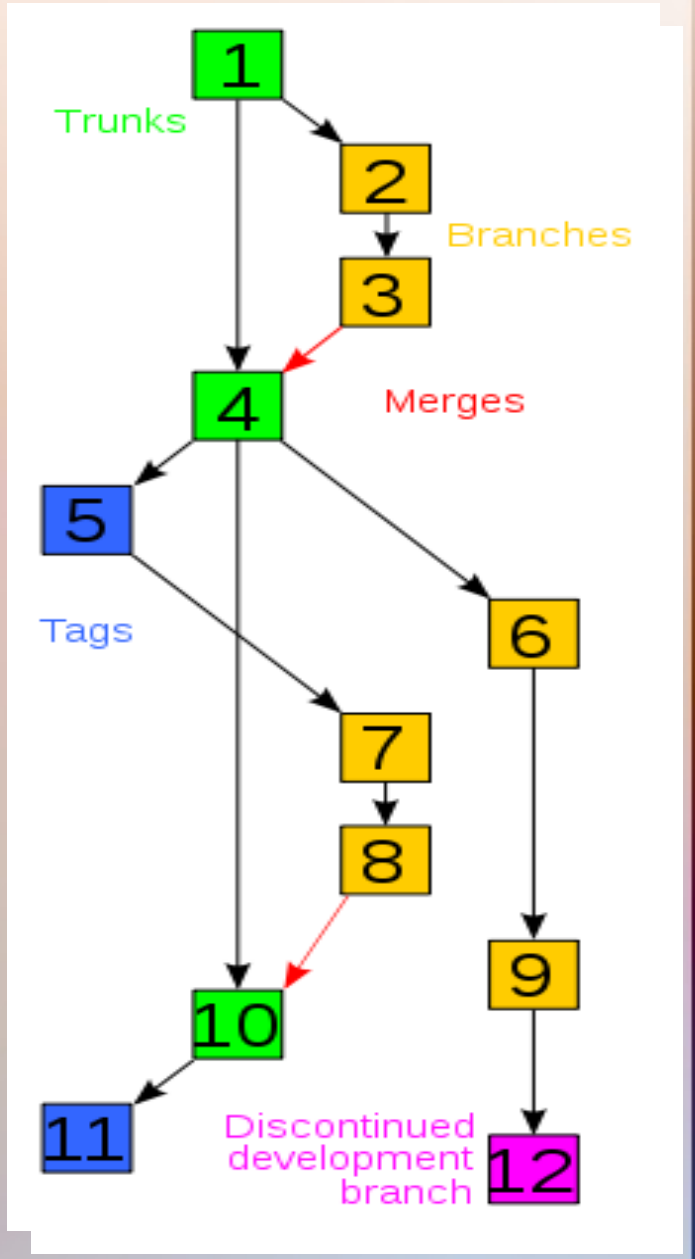


```
$ git branch -m master mymaster

$ git branch
* mymaster


$ git branch -m mymaster master
```

- The first has three parameters. The -m parameter tells Git to perform a move. The other two parameters are the old branch name and the new name you want your branch to have.
- The second command is 'git branch' by itself and displays the names of all the local branches in your repository.
- The third command renames mymaster back to master, just to keep everything consistent.



Naming Nodes by Their Hashes

Every document has a 40 character "signature" (hash) SHA-1 computed from all of its characters. It looks like:

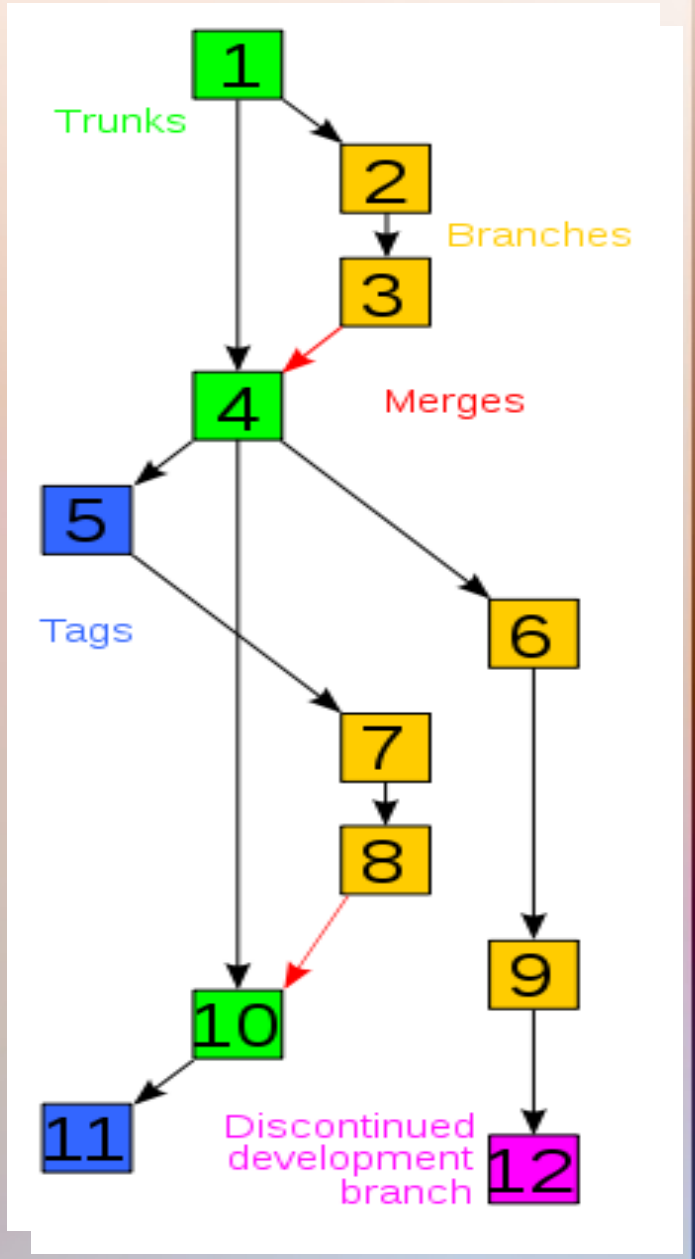
24b9da6552252987aa493b52f8696cd6d3b00373

The hash can be used as a “pointer” to locate its content

Identical files have the same hash and are represented by a single blob

View SHA1 of a commit:

```
$git rev-list HEAD^..HEAD
```



View Logs

- $ git log
- $ git log HEAD~4..HEAD”
- $ git log –pretty=oneline v1.0..v2.0 | wc-l
- $ git log --raw -r --abbrev=40 –pretty=oneline origin..HEAD
- $ git archive --format=tar --prefix=project/ HEAD | gzip >latest.tar.gz
- $ git blame <filename>

Add and Commit Review

Edit some files

Look at the output of “git status”

Ask GIT to keep track new files: “git add .”

Create new files

$ git diff –cached

$ git add .

$ git diff HEAD

$ git commit -m 'first commit'

More about Commit

- Think of a commit as a snapshot of your project — code, files, everything — at a particular point in time. More accurately, after your first commit, each subsequent commit is only a snapshot of your changes. For code files, this means it only takes a snapshot of the lines of code that have changed. For everything else like music or image files, it saves a new copy of the file.
- The previous commit executes actions locally, meaning you still haven't done anything on GitHub yet.

More about Commitment

Oops – git reset

Reset the conflicted merge/commit

```
$ git reset –mixed <commit-id>
```

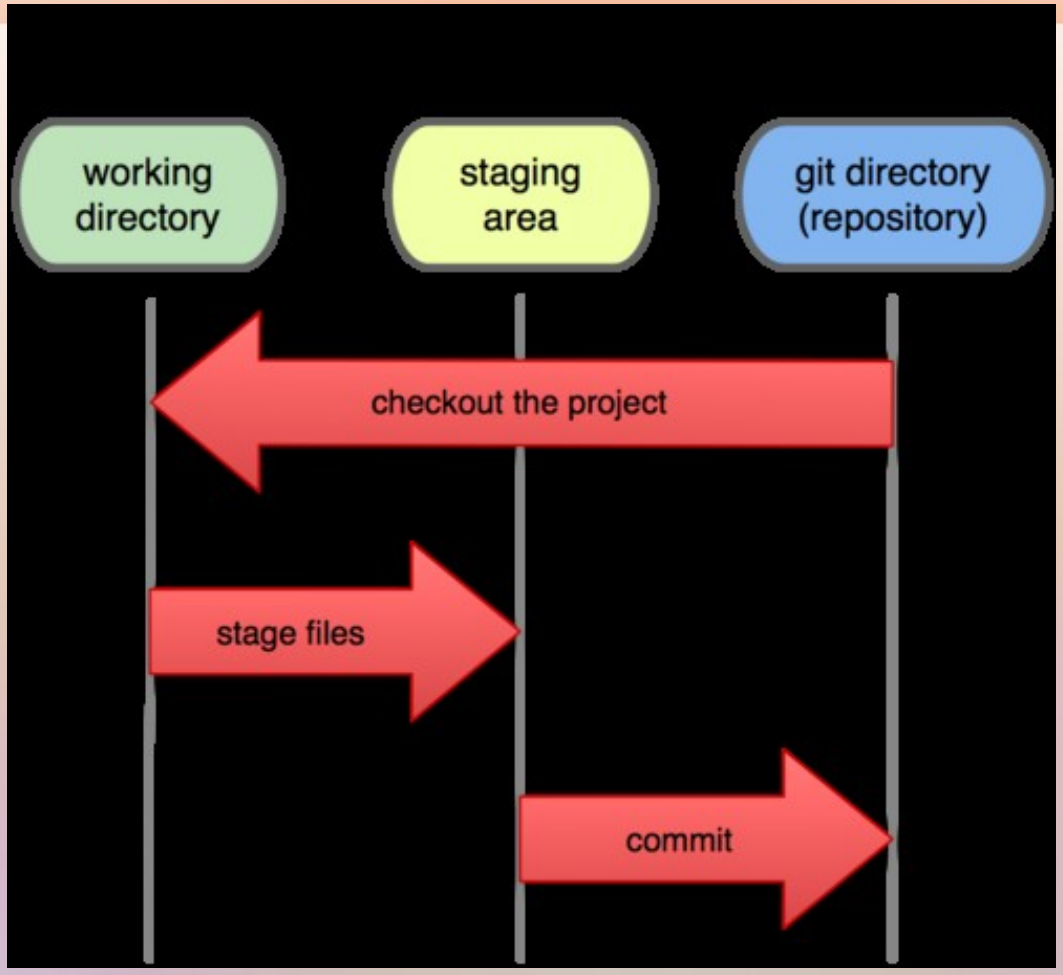
Reset the index database to the moment before merging

```
$ git reset –hard <commit-id>
```

Reset the index database and the working data

```
$ git reset –soft <commit-id>
```

You made a commit and also made some typo mistake. This commands help us fix those errors without touching the working data and index database



1. You modify files in your working directory.
2. You stage the files, adding snapshots of them to your staging area.
3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

Working with Remotes

- A remote is a repo stored on another computer, in this case on GitHub's server. It is standard practice (and also the default in some cases) to give the name origin to the remote that points to your main offsite repo (for example, your GitHub repo).
- $ git remote -v
- origin git://github.com/<username>/<projectName>.git

Push changes to Remote Repo

- To connect your local repository to your GitHub account, you will need to set a remote for your repo and push your commits to it:

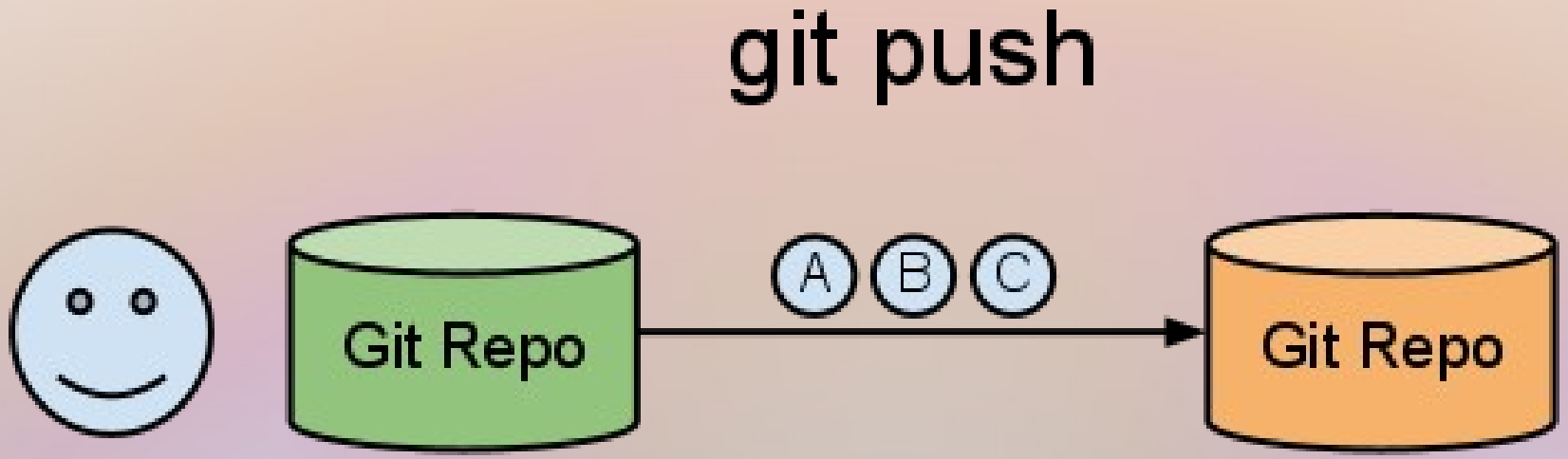
```
$ git remote add origin git@github.com:username/Hello-World.git
$ git push -u origin master
$ git remote -v
```

Create initial Repository (remote)

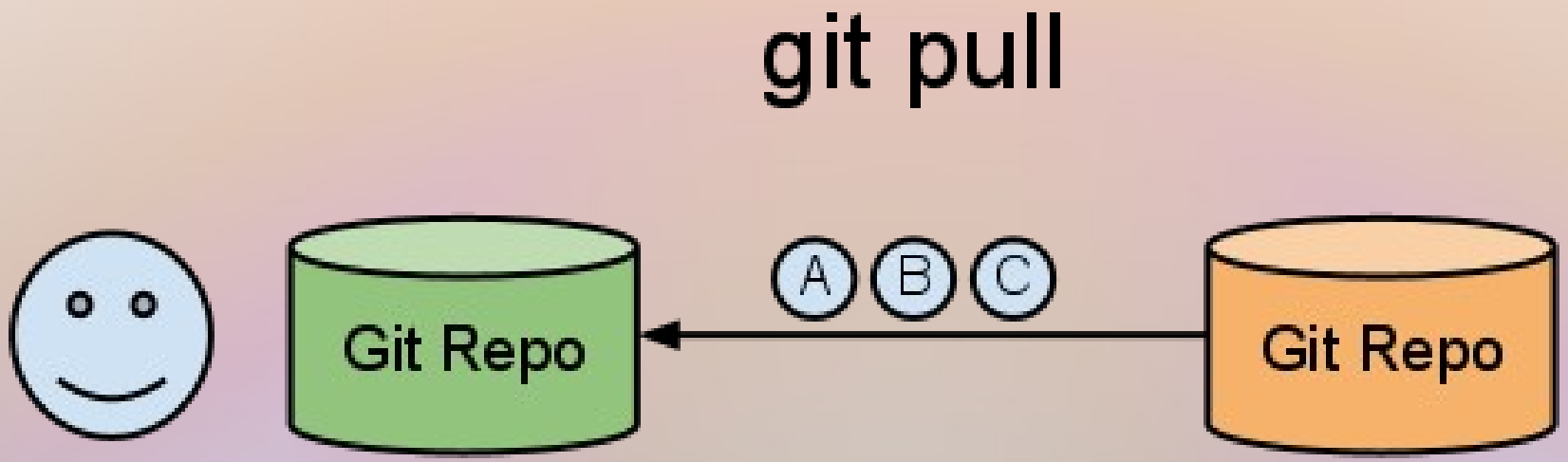
- $ `git remote add origin git@github.com:DallasHandsOnJava/dhoj01.git`
  [git status **and** git remote –verbose]
-
- OPTIONAL: cat .git/config
-
- $ `git push -u origin master`
  [git status]

- Reminder: a remote is a repository (repo) stored on another computer

Initial git push



git push has a related pull



Next Level

Git init

vs.

Git clone

Clone an existing Repository

Is it really a Bad Thing?

Clone an existing Repository

Clone an existing Repository

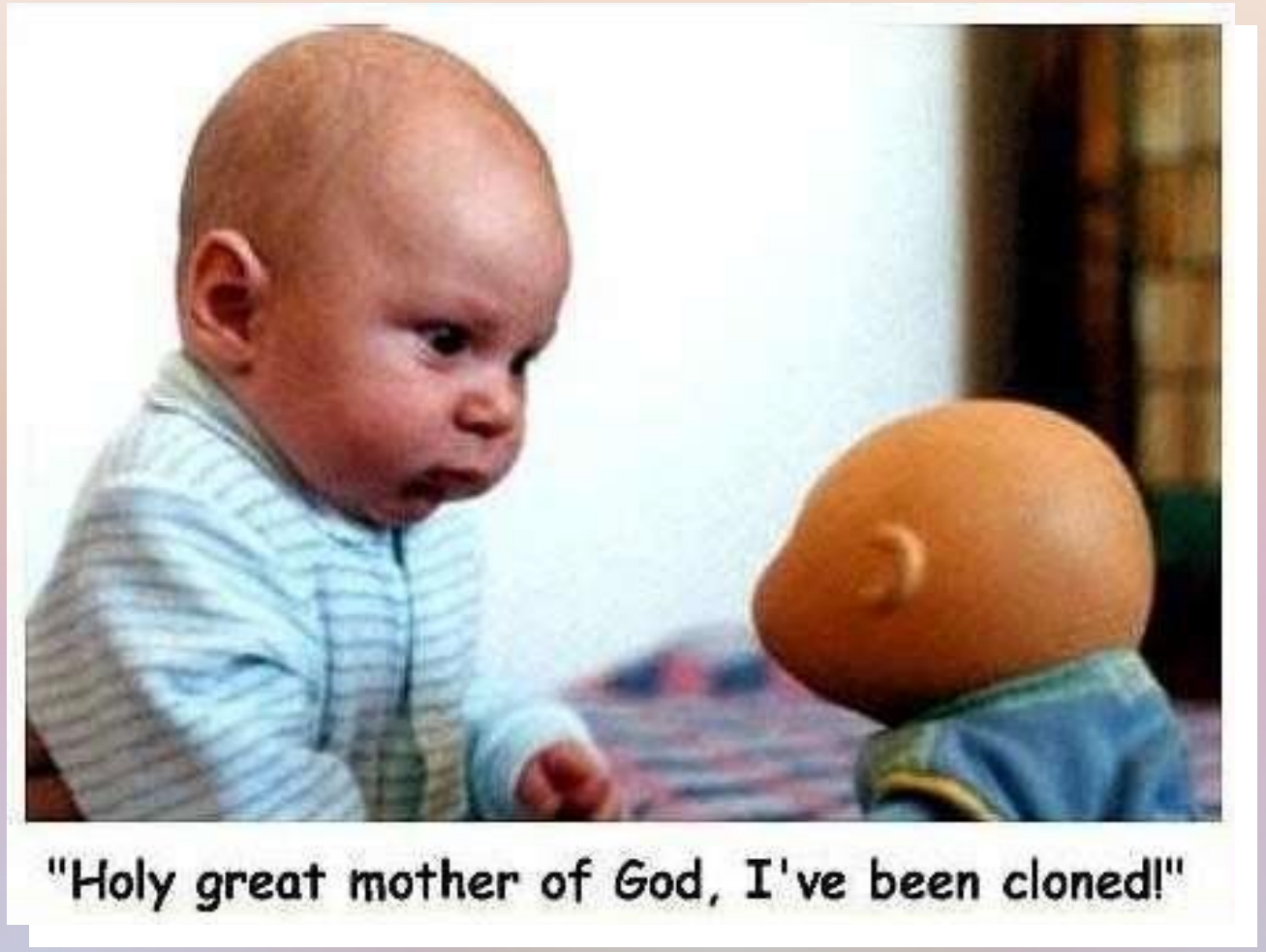
If you want to get a copy of an existing Git repository the command you need is **git clone**.

- Create a local repo (and optionally rename it):

```
$ cd /projects
$ git clone git@github.com:<username>/<projectName>
$ git clone git@github.com:DallasHandsOnJava/dhoj01 dhoj01_clone (1)
$ git clone git:/u/var/git/<projectName>/mysite.git mysite-clone (2)

(1) for a remote site from another host
(2) for a repository on a network drive
```

Clone an existing Repository



Let's look at the results.

Got all that ?

- git init
- git clone
- git add
- git commit
- git status
- git pull
- …

LET's Review and Play with it – (git and svn? Time Check)

Review and Play - Key Git Operations

1) **Init**. Create an empty Git repository

2) **Clone**. Copy a repository into a new directory. After cloning, edit, create and remove files for new version

3) **Add**. Add file contents from work space to the *index*. (The files are edited locally)

4) **Remove** = rm. Remove files from work space and from the *index*

5) **Commit**. Store the changes (that are added) to the repository, using the *index*. Completes this version.

6) **Branch**. Create (or delete) a branch

7) **Merge**. Join two or more branches

8) **Rebase**. Combine/restructure a set of commits to simplify them

9) **Checkout**. Checkout files etc from a commit, and switch work space to that new branch

10) **Fetch**. Download objects and refs from another repository

11) **Pull**. Fetch from and merge with another repository or a local branch

12) **Push**. Update remote refs (in another repo) along with associated objects

Where R We ? Q &A


Found balance
Bit addicted
What The? Totally confused!
HELP
Found my life's purpose
I'm not getting it at all!
I'm starting to understand!
Understanding
Effort

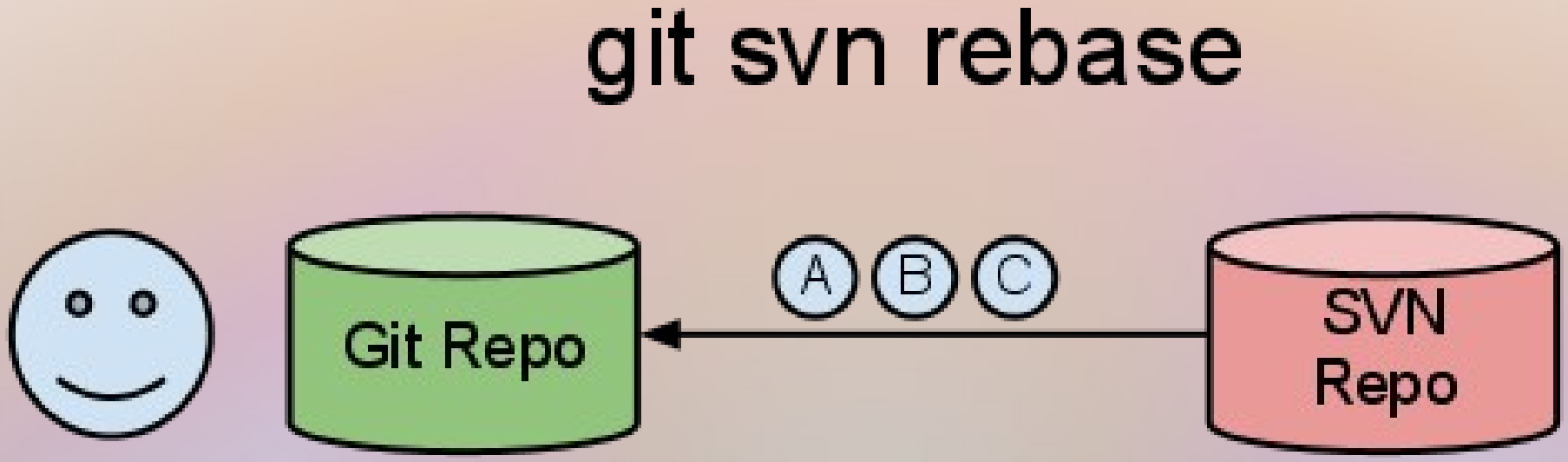
Migrating from/using SVN ?

- git-svn allows you to use git on your local machine (even offline) and push your commits to yoursubversion repository
- Preform normal git commits and branches
- git svn rebase to “pull” latest updates
- git svn dcommit to “push” your local commits

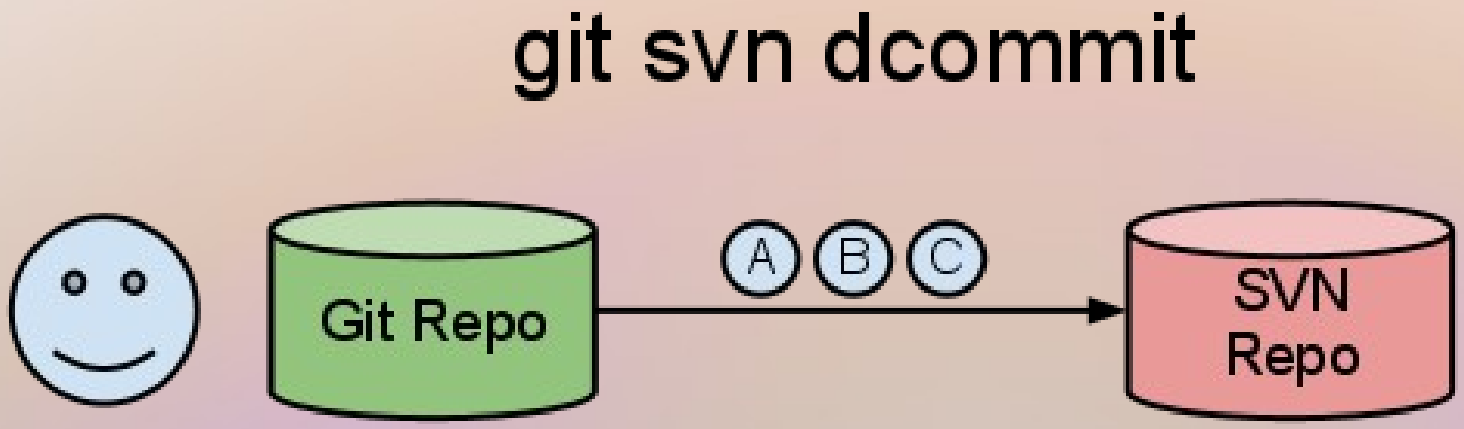
How to Use git-svn

- Checkout Latest Trunk
- `$ git svn init https://svn.jboss.org/repos/maven/demo/trunk`
- `$ git svn fetch -r558`
-
- Convert the SVN History
- `$ git svn clone https://svn.jboss.org/repos/maven/demo/`

Git svn rebase



git svn dcommit



Thank You

Credits / Sources

History of Revision Control

- http://en.wikipedia.org/wiki/Revision_control
- http://code.google.com/p/pysync/wiki/VCSHistory
- http://netsplit.com/2007/09/26/why-i-choose-bazaar-a-history-of-revision-control/
- http://www.catb.org/~esr/writings/version-control/version-control.html

Credits / Sources

Free Book / Documentation on Git

- http://progit.org/book/
- http://git-scm.com/documentation
- Intro to Git http://learn.github.com/p/intro.html
- http://www.cheat-sheets.org/#Git
- Everyday Git in 20 commands: http://schacon.github.com/git/everyday.html
- http://trac.parrot.org/parrot/wiki/git-svn-tutorial

Credits / Sources

Acquire Git

- http://code.google.com/p/msysgit/ Git for Windows
- https://github.com/msysgit/msysgit/wiki/
- Download GIT : http://git-scm.com/download
- for WinTel http://msysgit.googlecode.com/.

Credits / Sources

Git Server

- Install (Set Up) Git and Gitosis on Ubuntu http://blog.agdunn.net/?p=277
- How To: Install and Configure GitWeb http://gofedora.com/how-to-install-configure-gitweb/

- Who
- What
- Where
- When
- Why
- How

- Who
- What
- Where
- When
- Why
- How