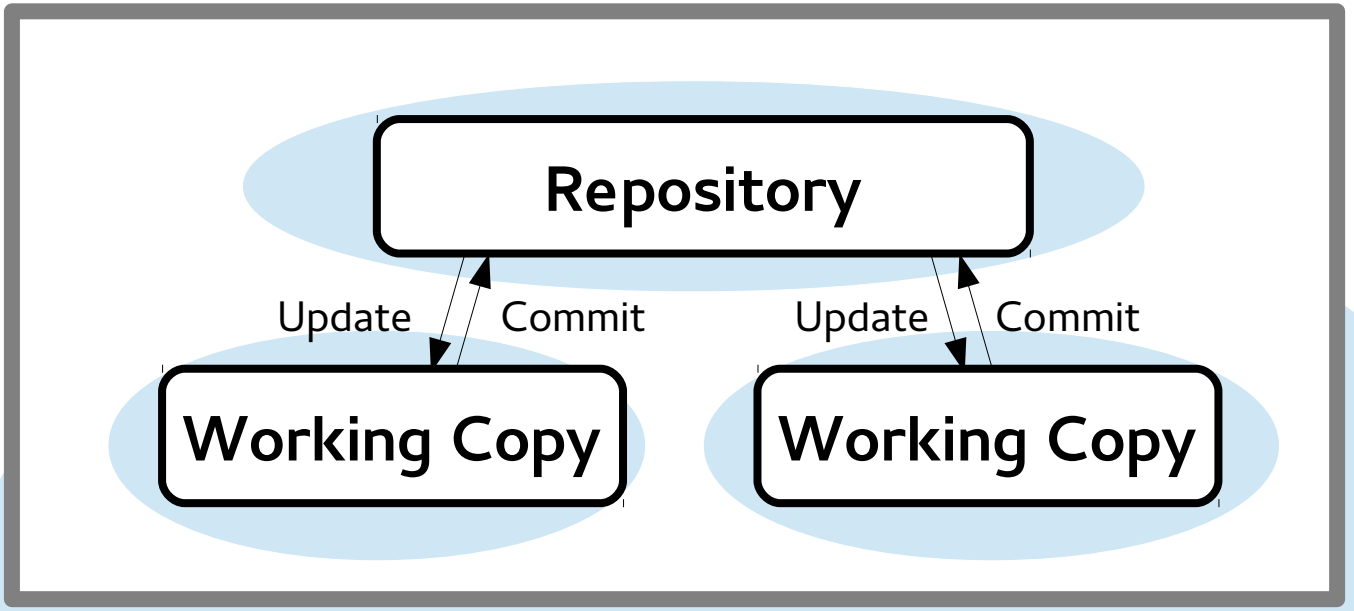
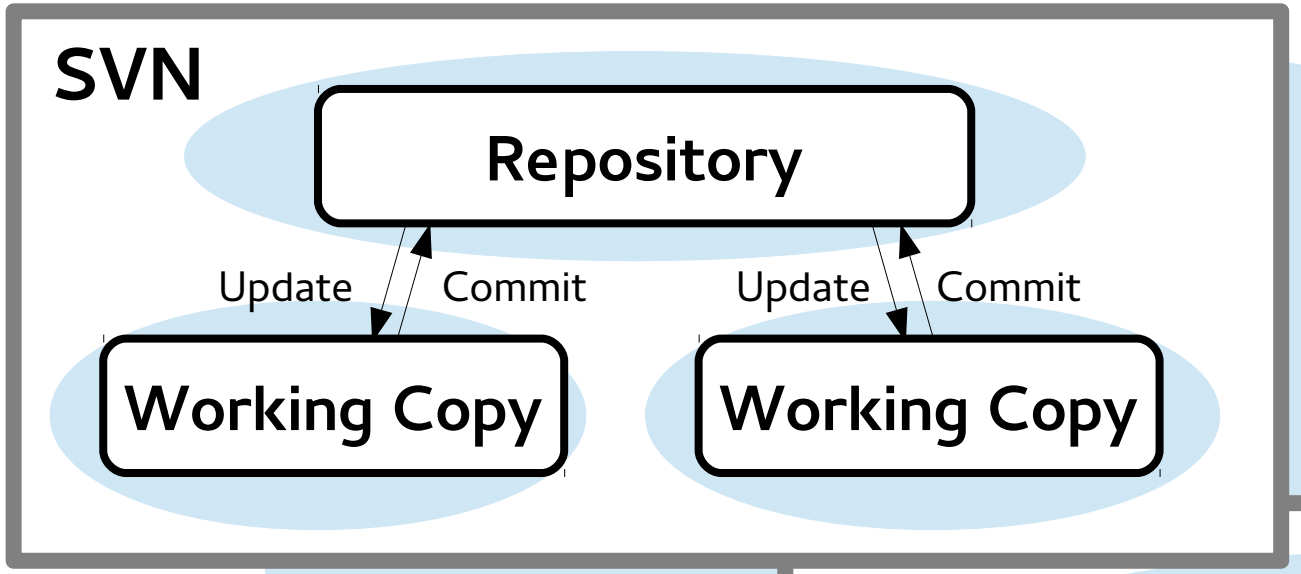
# Git vs SVN

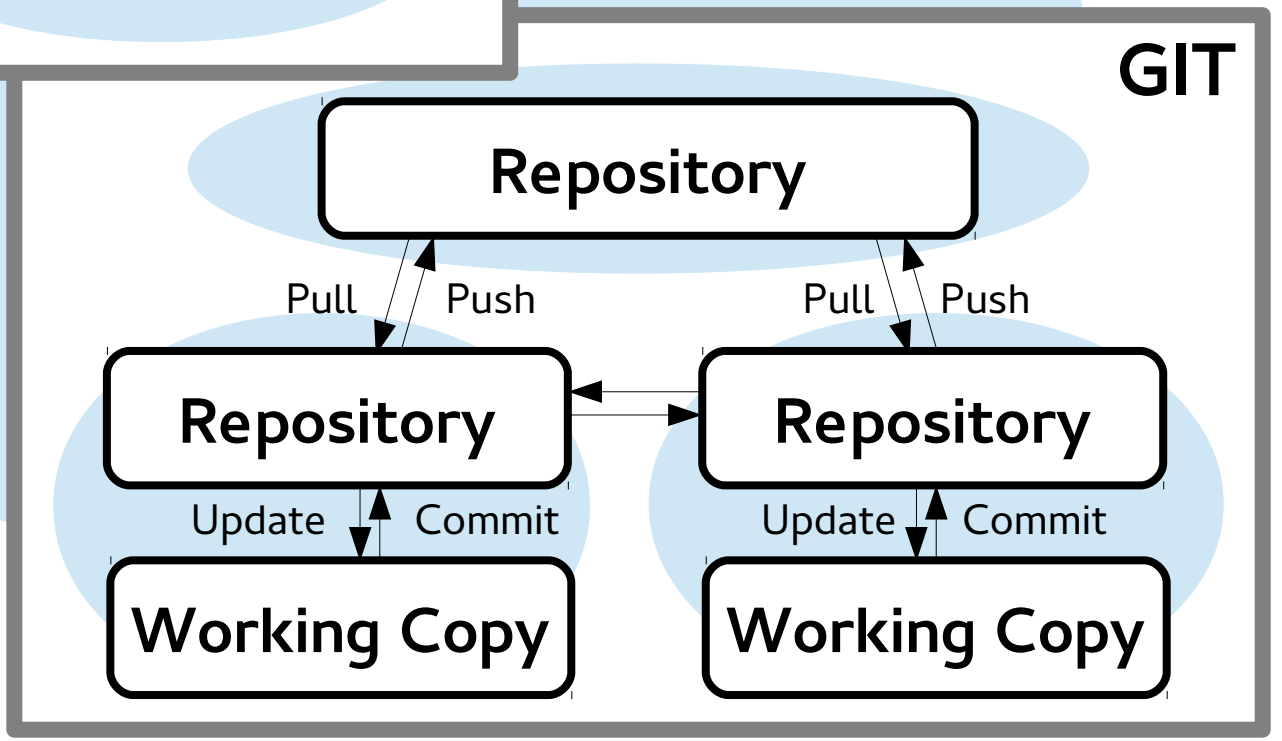
# What is SVN?




- Centralised Version Control
- One big remote repository
- Checkout a branch from this central repository
- Commit connects to remote and sends changes
- Improved on CVS, similar concepts

# Comparing Git to Svn




- Distributed Version Control
- "Clone" complete copies of the entire repository
- "Commit" stores local snapshot of working index
- Push and pull to any other "remote" git repository

# Checking out a Repo

- Svn checkout:
  - Makes a **local copy of the tree** in a repository and matches each file to a remote one
  - Can **checkout a sub-directory** of a repository
  - Every directory has a `.svn/` directory
- Git clone:
  - Makes a **local copy of the repository** and makes your working index match the head of the master branch
  - Can only **check-out an entire repository** *(sort of)*
  - Top-level directory will contain a `.git/` directory

# Commands are changed

| SVN | GIT |
|---|---|
| checkout *repository* | clone *repository* |
| checkout *sub-directory* | *Not possible* |
| commit | commit + push |
| revert *filename* | checkout *filename* |
| switch *branch* | checkout *branch* |
| update | pull |
| export | *Sparse clones but not so simple* |
| add *filename* | add *filename* |
| Log / status / diff / blame | Log / status / diff / blame |

# Resetting the Working Copy

- Having made some changes, we want to roll them back
- In SVN:

```
$ svn revert -R directory/
Reverted 'directory/file1'
Reverted 'directory/file2'

$ svn revert filename
Reverted 'filename'
```

- In Git, it depends whether we have changed:
  - Working index:

```
$ git checkout filename
$ git checkout directory/
```

  - Staging area (after `` `git add` ``):

```
$ git reset filename
Unstaged changes after reset:
M       filename
```

# Merge Resolution

**File conflicts:**

- User A and B edit same file in the same place
  - Svn and git need to manually merge files
- Working with the merge interactively:
  - Svn gives you options immediately
  - Git will return control to you immediately
    - Use `git mergetool` which will give a more interactive (even GUI, if configured) tool

# Merge Resolution

## File conflicts:

- Finishing merges

```
$ svn update
Conflict discovered in 'file1'.
Select: (p) postpone, (df) diff-full, (e) edit,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options: p

$ vi file1 # or emacs, sublime etc
.....
<<<<<<< .mine
changes by user1
=======
changes by user2
>>>>>>> .r2
.....
# Select desired hunk

$ svn resolve --accept working file1
$ svn commit -m "Fixed conflict"
```

```
$ git pull
Auto-merging file1
CONFLICT (content): Merge conflict in file1
Automatic merge failed; fix conflicts and then comm
it the result.

$ vi file1 # or emacs, sublime etc
.....
<<<<<<< HEAD
changes by user1
=======
changes by user2
>>>>>>> branch1
.....
# Select desired hunk

$ git add file1
$ git commit -m "Fixed conflict"
```

- Switch file versions:

```
$ git checkout --theirs filename
$ git checkout --ours filename
```

- Abort merge:

```
$ git merge --abort
```

# Merge Resolution

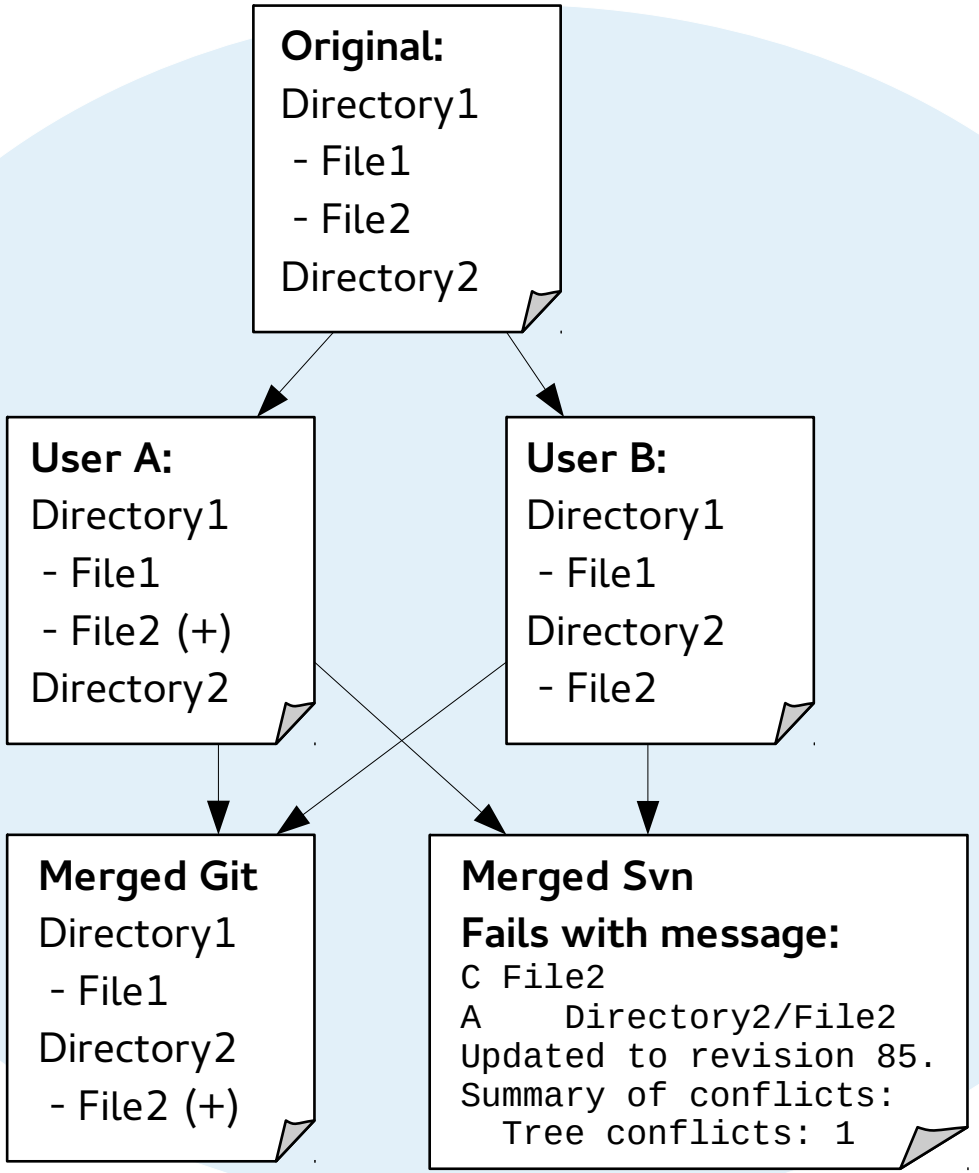
**Merging Gotchas**

- --theirs is the incoming file
- --ours is the current file
  - So when Merging, 'theirs' is the branch being merged in, 'ours' is the branch being merged into.
  - When rebasing, 'ours' is the commits being rebased onto (typically the remote, the other branch), 'theirs' is the branch being rebased (the branch being worked on).
- Use `` `git log --merge -p filename` `` to look at changes to a file that contribute to a conflict
- `` `Git merge branch2` `` will merge branch2 into your current branch

# Merge Resolution

## Tree Conflicts

- User A renames or moves a file (even to a sub-dir)
- User B changes its content
- Git can resolve automatically
- Svn will flag as a conflict
  - Need to solve manually



# Tagging a Release

- Repository IDS
  - SVN revision numbers: `r1401`
  - Git commit hashes `ff9e41983dd160cdc20d048a4153fa49c37a1b8f`
- Specific tags emphasize a release:
  - In SVN: Copy the trunk into the tags directory

```
$ svn copy http://svn.example.com/repos/calc/trunk \
           http://svn.example.com/repos/calc/tags/release-1.0 \
           -m "Tagging the 1.0 release of the 'calc' project."

Committed revision 902.
```

  - In Git: Use `` `git tag` ``

```
$ git tag release-1.0
$ git tag -a release-1.1 -m "This is a new release"
```

# Git Spice:

## Some additional techniques

# Git Config

- Git config controls setup
  - --system – All users ( /etc/gitconfig )
  - --global – All your repositories ( ~/.gitconfig )
  - --local – Just the current repository ( aProject/.git/config )
- Email and username

```
$ git config --global user.name "Ben Krikler"
$ git config --global user.email bek07@ic.ac.uk
```

- Colour
  - Switches on colour in diffs, logs, status etc
  - Enabled by default in recent Git versions

```
$ git config --global color.ui true
```

- More at: http://www.git-scm.com/book/en/v2/Customizing-Git-Git-Configuration

# Git Alias

- Work like bash aliases
- Make an 'unstage' command

```
$ git config --global alias.unstage 'reset HEAD --'
$ git unstage
Unstaged changes after reset:
M       snippets.txt
```

- Different log output:

```
$ git config  --global  alias.lg  "log --graph
     --abbrev-commit --date=relative
     --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset \
 %s %Cgreen(%cr)  %C(bold blue)<%an>%Creset'"
```

```
1 ben@bens-laptop: ben/alcap/AlcapDAQ 19:24:16$ git lg
* 87467b4 - (HEAD, origin/develop, origin/HEAD, develop)  Fixed Ge efficiency calculation. Script printou
* 40c1f20 -  Added a GetEnergy method to TDPs (7 weeks ago)  <Ben Krikler>
*   a5fa1ad -  Merge branch 'feature/BK_Si_calibration' into develop (7 weeks ago)  <Ben Krikler>
|\
| * a258f86 -  Remove the TDiff methods from production.cfg since we're not needing this now (7 weeks ag
| * 6538bc5 -  Forgot to add new implementation for TVAnalysedPulseGenerator (7 weeks ago)  <Ben Krikler>
| * cb78f6c -  Add calibration method to Generators but implement default in base class (7 weeks ago)  <B
| * e623993 -  Make sure SetupNavigator still compiles (7 weeks ago)  <Ben Krikler>
| * b58dae9 -  Set the BankName within TVAnalysedPulseGenerator and keep it (7 weeks ago)  <Ben Krikler>
| * 9b93d68 -  Tidy up the SetupNavigator a little for Energy calibration (7 weeks ago)  <Ben Krikler>
|/
* 083b729 -  Make sure we add Debugging to the generators if MakeAnalysedPulses is run with the 'debug'
*   0688084 -  Merge branch 'feature/BK_mu_stops' into develop (7 weeks ago)  <Ben Krikler>
|\
```

# Ingoring files

- Why and when to use:
  - Ignore a set of files or a directory
  - Eg. Emacs back-up files shouldn't be committed
- How to use:
  - Write a .gitignore file in the directory containing files to be ignored
  - In the file:
    - Comment lines start with `#`
    - Wildcard with `*`
    - Character sets such as `[abc]`, `[a-z]`
    - Extended globbing (like bash) so `**` matches across directories
    - Negate a match by prefixing `!`
- Many standard `.gitignore` files can be found at:
  - https://github.com/github/gitignore

# Ingoring files

- Example .gitignore: c++.gitignore
  - from https://github.com/github/gitignore

```
 1 # Compiled Object files
 2 *.slo
 3 *.lo
 4 *.o
 5 *.obj
 6 # Precompiled Headers
 7 *.gch
 8 *.pch
 9 # Compiled Dynamic libraries
10 *.so
11 *.dylib
12 *.dll
13 # Fortran module files
14 *.mod
15 # Compiled Static libraries
16 *.lai
17 *.la
18 *.a
19 *.lib
20 # Executables
21 *.exe
22 *.out
23 *.app
```

# Sparse Repository

- Why and when to use:
  - Want a sub-directory of a git repo
- How to use:
  - Follow guide here:
  - briancoyner.github.io/blog/2013/06/05/git-sparse-checkout/

# Git commit --amend

- Why and When:
  - Wish to change the commit message on the previous commit
- How:

```
$ git commit --amend -m "This is the new commit message"
$ git commit --amend -F message.txt
$ git commit --amend
```

  - Set the `EDITOR` environment variable in the shell for the last command to open the commit message in your preffered editor (eg. Vim)
  - Warning: Don't amend commits that have been pushed!!

# Git blame

- Why and When:
  - Find out last person to touch each line of code
- How:

```
$ git blame filename
$ git blame -MC filename
```

  - `-MC` Shows the original file if the line is from another file that changed in the same commit
- Output:

```
30656cc7 (benkrikler        2013-06-16 01:25:53 +0100  1) Example-Makefiles
dae9f6c6 (Chris Hunt        2013-04-20 23:41:26 +0200  2) =============
dae9f6c6 (Chris Hunt        2013-04-20 23:41:26 +0200  3)
30656cc7 (benkrikler        2013-06-16 01:25:53 +0100  4) Two makefiles that build either an executa
dae9f6c6 (Chris Hunt        2013-04-20 23:41:26 +0200  5)
30656cc7 (benkrikler        2013-06-16 01:25:53 +0100  6) Assumes that every header file is containe
30656cc7 (benkrikler        2013-06-16 01:25:53 +0100  7) All implementation files must be in ```src
96360e79 (Christopher Hunt  2013-04-24 15:53:32 +0100  8)
30656cc7 (benkrikler        2013-06-16 01:25:53 +0100  9) This can be customised by changing the va
30656cc7 (benkrikler        2013-06-16 01:25:53 +0100 10)
30656cc7 (benkrikler        2013-06-16 01:25:53 +0100 11) Usage
30656cc7 (benkrikler        2013-06-16 01:25:53 +0100 12) ----
76d70889 (benkrikler        2013-06-18 12:34:48 +0200 13) Type ```make``` and all executables will
```

# Git Stash

- Why and when to use:
  - To quickly strip away changes to a working index
  - When you wish to switch a branch but aren't ready to commit some local change
- Has separate sub-commands:
  - Git stash [save] – Stash away local changes
  - Git stash apply – Apply the latest stash to the working index
  - Git stash pop – Apply then remove the latest stash
  - Git stash list – List all available stashes and their hashes
  - Git stash drop – Remove a stash from the list
  - Git stash show – Show what changes the stash represents
  - Git stash branch – Turn the stash into a new branch
- Links:
  - http://www.git-scm.com/book/en/v2/Git-Tools-Stashing-and-Cleaning

# Git Stash

- Make some local changes:

```
$ git status -s
 M filename
```

- Stash the changes:

```
$ git stash
Saved working directory and index state WIP on master: ff9e419 Add all text files
HEAD is now at ff9e419 Add all text files
```

- Inspect the new stash:

```
$ git stash show
 snippets.txt | 20 ++++++++++++++++++++
 1 file changed, 20 insertions(+)
$ git stash show --full-diff
diff --git a/snippets.txt b/snippets.txt
index 0ce3094..af4cb46 100644
--- a/snippets.txt
+++ b/snippets.txt
@@ -1,8 +1,9 @@
 something something
+
-something or something
+something or nothing
```

# Git Stash

- List available stashes

```
$ git stash list
stash@{1}: WIP on master: ff9e419 Add all text files
stash@{0}: WIP on master: ff9e419 Add all text files
```

- Pop the last stash

```
$ git stash pop
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 working directory)

        modified:   snippets.txt
$ git stash list
stash@{0}: WIP on master: ff9e419 Add all text files
```

- Delete the remaining stash

```
$ git stash drop
Dropped refs/stash@{0} (141f59dea279c603a1afefa6ad5e1094c16bebab)
```

# Git Bisect

- Why and when to use:
  - Identify the commit where a bug or change was introduced
- What:
  - Performs a binary search through commits until you identify the change
- How:
  - Identify the range of commits to inspect
  - Setup git bisect
  - Git takes you to the mid-point of your range
  - Inspect this commit (compile, run, debug etc)
  - Tell git bisect if this commit is good or bad
  - Repeat the last three points until you find the culprit commit
- Links:
  - https://www.kernel.org/pub/software/scm/git/docs/git-bisect.html

# Git Bisect

- Find the range to inspect (git log)
- Setup:

```
$ git bisect start
$ git bisect bad                 # Current version is bad
$ git bisect good v2.6.13-rc2    # v2.6.13-rc2 was the last version
                                 # tested that was good
Bisecting: 675 revisions left to test after this
```

- Test this commit
- Tell git the result:

```
$ git bisect good                          # this one is good
Bisecting: 337 revisions left to test after this
```

- Also:
  - `git bisect reset`: Return to the original state
  - `git bisect skip`: Test a different commit nearby
  - `git bisect run my_script arguments`: Automate everything

# Git Cherry-pick

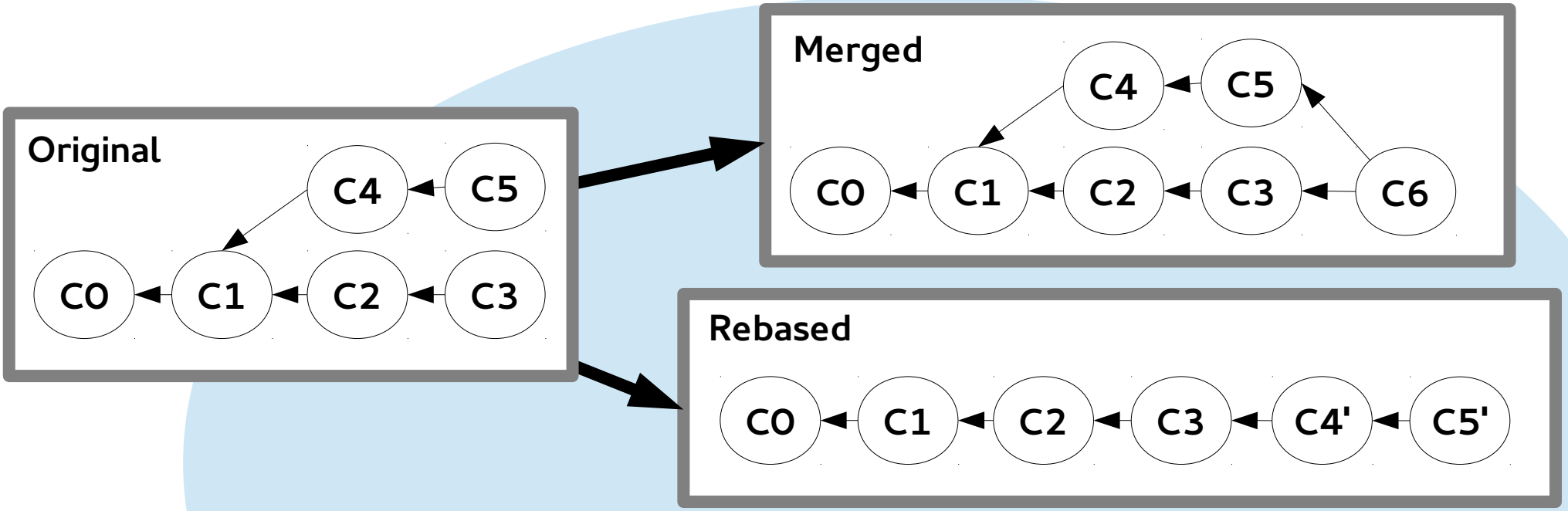
- Why and when to use:
  - Apply commits from another branch selectively
  - Contrast to merge / rebase
- How:
  - Find commits of interest
  - Change to receiving branch
  - Run: `git cherry-pick` *`commit_hash`*

# Git Workflows

# Work Flows

- Git allows for:
  - Multiple remote repositories
  - Easy branching / merging (on the whole)
- Rebase vs Merge
- Schemas:
  - Centralised
  - Integration Manager (Esp. GitHub, CMS)
  - Dictator vs Lieutenant
- Git-flow

# Rebase Vs Merge

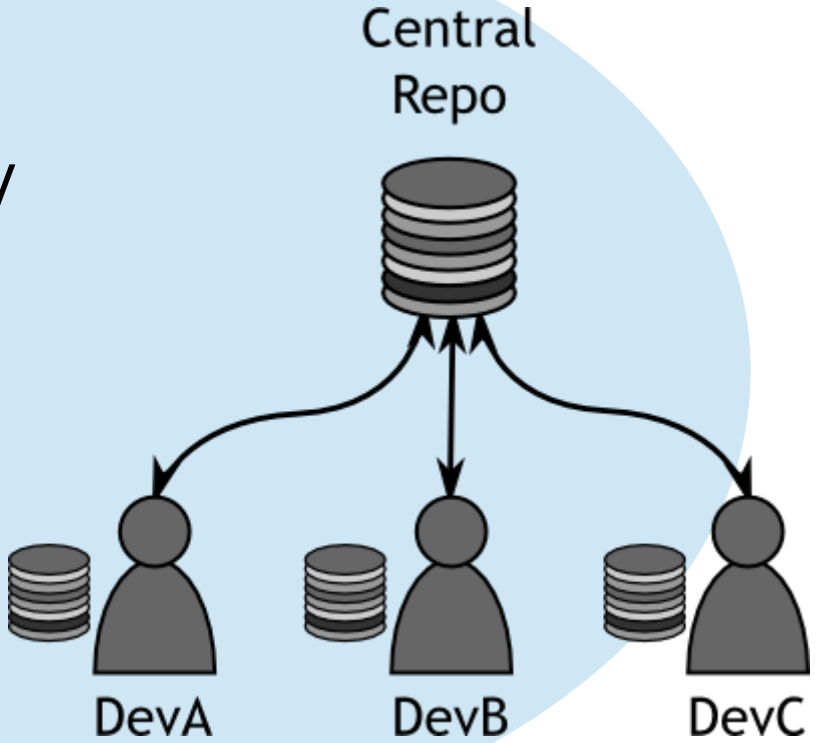



- Some groups state rebasing as preferred
- Rebasing 'linearises' the history
  - Can become easier to read
  - Avoid rebasing if the branch is public
  - If the branch history is important

# Collaboration Schemes

## Centralised Organisation

- Everyone push and pull freely
- Single central remote
- Better for smaller groups
- Essentially the SVN model

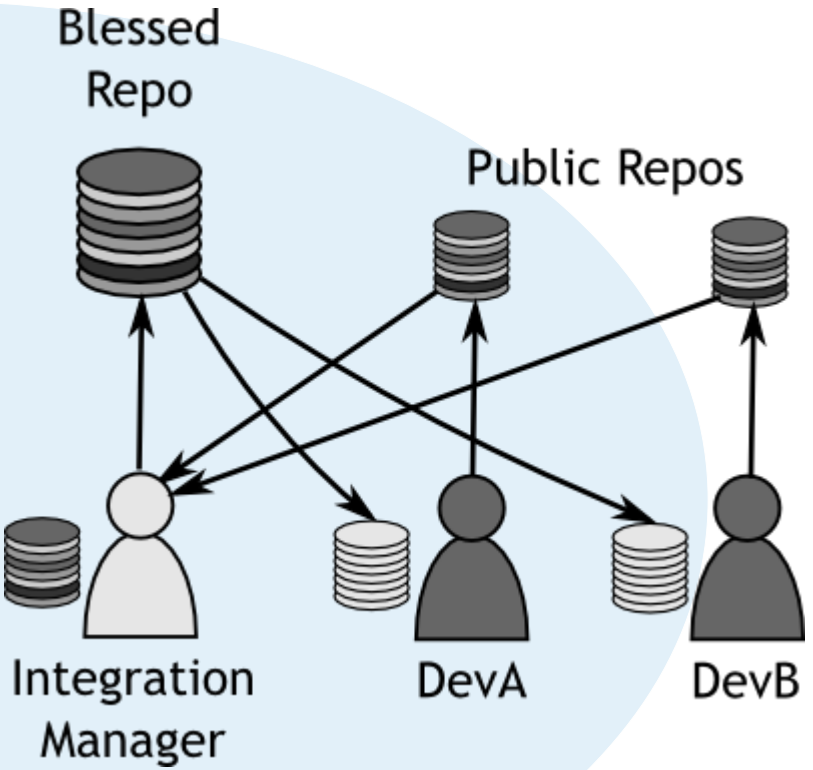



Images from cbx33.github.io/gitt/chap1-2.html

# Collaboration Schemes

## Integration Manager

- Each developer has a private (local) and public (eg. GitHub) repository
- Integration pulls on request from public repos
- Developers rebase on blessed repo
- Quite common
  - CMS use this approach

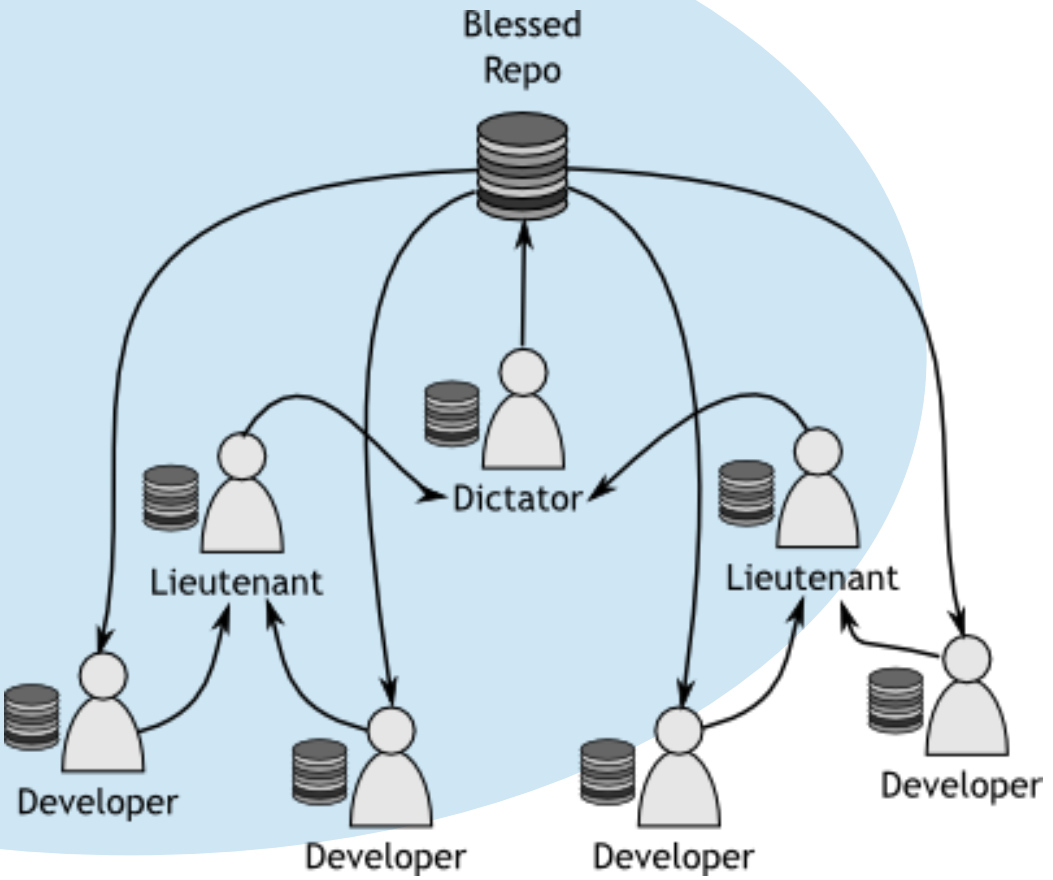



Images from cbx33.github.io/gitt/chap1-2.html

# Collaboration Schemes

## Dictator vs Lieutenant

- Delegation of merging
- Less common
- Used in very big projects
  - Eg. Linux Kernel




Images from cbx33.github.io/gitt/chap1-2.html

# Git flow

- Prescription for branching
- Can be used for collaboration
  - Normally with centralised setup
- Git add-on to help manage branching:
  - github.com/nvie/gitflow

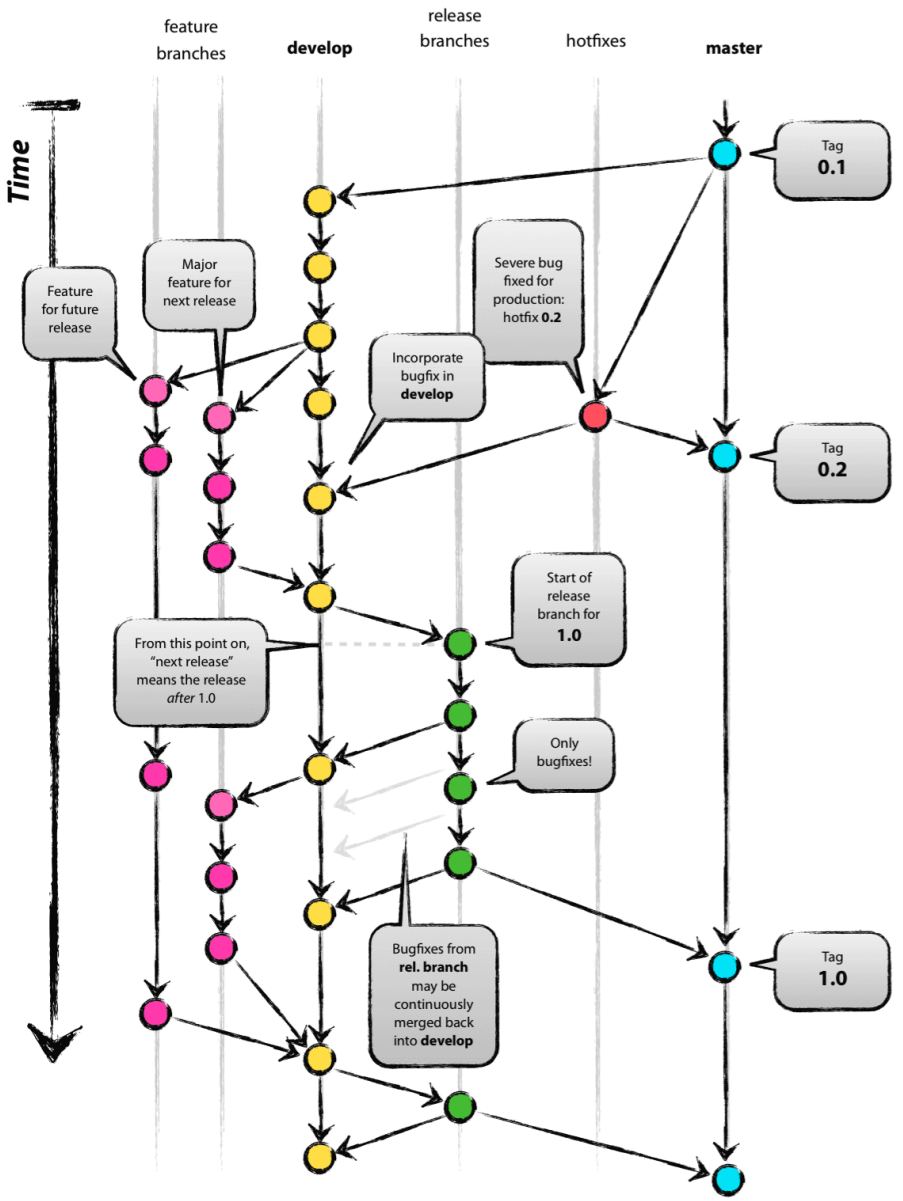



Image from nvie.com/posts/a-successful-git-branching-model/

# Summary

- Git is very powerful
- Git has many tools and approaches
- Git is very different to CVS and SVN
- Git
- Git
- Giiiit
- They're Giiiiiiiiiit (read as Tony the Tiger)
- Git off my land

# Links

- Kick-ass interactive cheat-sheet:
  - ndpsoftware.com/git-cheatsheet.html
- Nice guidelines and tutorial:
  - cbx33.github.io/gitt/intro.html
- Github + CodeSchool's 15 min git walkthrough
  - try.github.io/levels/1/challenges/1
- Working with Github
  - guides.github.com/