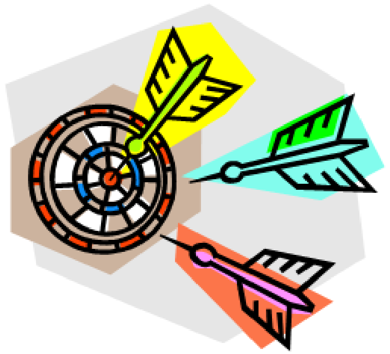
Mercurial I

Developed by
Jeppesen Crew Academy

Course goals

This course will teach you

- Common Mercurial commands
- Useful Mercurial Extensions
- Practical Concepts
- Where to find Help Resources



Prerequisites

- Basic Linux command line knowledge

Please –

Don't be afraid to ask questions if anything is unclear!

Agenda

09:00 - 10:15	Introduction, core concepts
10:15 - 10:30	**Coffee break**
10:30 - 12:30	Basic Mercurial usage
12:30 - 13:30	**Lunch**
13:30 - 15:00	Getting more advanced
15:00 - 15:15	**Coffee break**
15:15 - 17:00	Final session

All times are approximate – changes may/will occur

Short breaks every ~40 minutes or so

Chapter 1 – Introduction

- What is Version Control?
- Distributed Version Control
- Core Concepts

Introduction

- A Version Control System, or VCS, stores complete version and change information about individual and/or a collection of files
- Provides ability to track changes over time, with the possibility to revert to a particular revision
- Metadata used to link developer’s working area with repository

Introduction

Centralized Version Control

- One central repository is used by all developers on a project

- Typically focused on changes at the individual file level
  - Suitable when programs were contained in single files (circa 1970s)

- Later versions added concurrency and remote repository control features

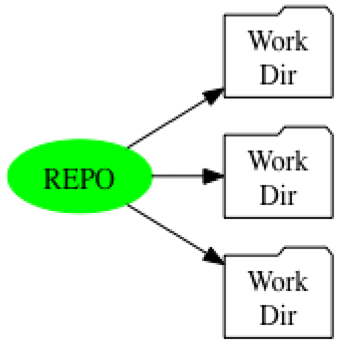
Introduction

Distributed Version Control

- One or many repositories used by developers on a project
- Each developer has a working clone repository
- Full metadata is distributed with each clone repository
- Connection to a central repository not required to capture and track local changes

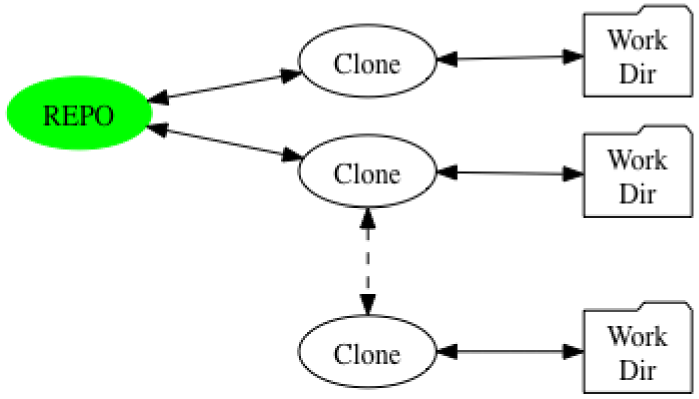
Introduction

Centralized



Changes cannot be shared directly between users

Distributed



Users can share changes directly without using a central repository

The Repository

The Repository is the central theme for Mercurial

- The repository for a project contains all of the files that “belong to” that project, along with a historical record of the project's files

- In practice, a repository is simply a collection of managed files in a browseable directory structure

- All repositories from the same family can exchange changesets and patchsets

The Clone

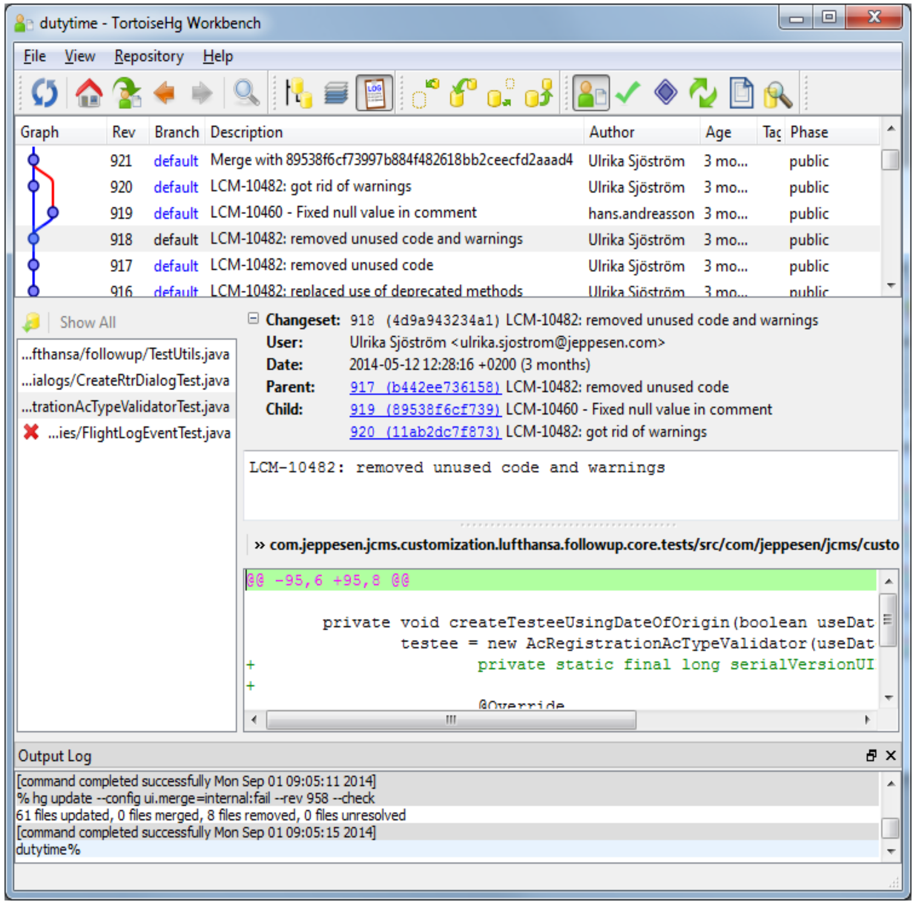
- As in Mercurial, the term clone is commonly defined as:
  - *A person or thing that duplicates, imitates or closely resembles another in appearance, function, performance or style*

- In Mercurial, a clone is an identical copy of the original (which might be itself a clone)

- All clones are themselves repositories

The Changeset

- A changeset is a collection of changes to one or many project files
- Natural for modern applications with a large set of inter-related source code files
- Easier to track down regression bugs
- By comparison, changes to files in CVS are tracked individually on a file-by-file basis.



Graph	Rev	Branch	Description	Author	Age	Tag	Phase
	921	default	Merge with 89538f6cf73997b884f482618bb2ceecfd2aaad4	Ulrika Sjöström	3 mo...		public
	920	default	LCM-10482: got rid of warnings	Ulrika Sjöström	3 mo...		public
	919	default	LCM-10460 - Fixed null value in comment	hans.andreasson	3 mo...		public
	918	default	LCM-10482: removed unused code and warnings	Ulrika Sjöström	3 mo...		public
	917	default	LCM-10482: removed unused code	Ulrika Sjöström	3 mo...		public
	916	default	LCM-10482: replaced use of deprecated methods	Ulrika Sjöström	3 mo...		public

...fthansa/followup/TestUtils.java
...ialogs/CreateRtrDialogTest.java
...trationAcTypeValidatorTest.java
...ies/FlightLogEventTest.java

Changeset: 918 (4d9a943234a1) LCM-10482: removed unused code and warnings
User: Ulrika Sjöström <ulrika.sjostrom@jeppesen.com>
Date: 2014-05-12 12:28:16 +0200 (3 months)
Parent: 917 (b442ee736158) LCM-10482: removed unused code
Child: 919 (89538f6cf739) LCM-10460 - Fixed null value in comment
920 (11ab2dc7f873) LCM-10482: got rid of warnings

```
LCM-10482: removed unused code and warnings
```

» com.jeppesen.jcms.customization.lufthansa.followup.core.tests/src/com/jeppesen/jcms/custo

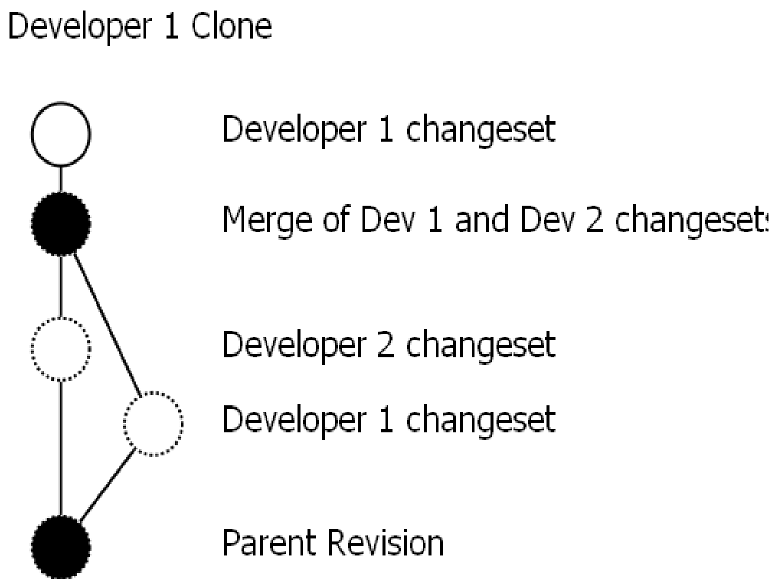
```
@@ -95,6 +95,8 @@

         private void createTesteeUsingDateOfOrigin(boolean useDat
                 testee = new AcRegistrationAcTypeValidator(useDat
+                        private static final long serialVersionUI
+
                        @Override
```

```
[command completed successfully Mon Sep 01 09:05:11 2014]
% hg update --config ui.merge=internal:fail --rev 958 --check
61 files updated, 0 files merged, 8 files removed, 0 files unresolved
[command completed successfully Mon Sep 01 09:05:15 2014]
dutytime%
```

Implicit Branching

- A branch is created when two changesets are created independently of each other
- All changesets are implicit branches
- Merging is a merge of changesets' revisions to a common base, including conflicting changes on a file basis



The Mercurial Command Line

Mercurial is invoked from the command line,

it uses a standard syntax:

`hg [opts] <command> [params]`

Common `opts`:

`-q,--quiet`	suppress output
`-v,--verbose`	enable additional output
`--config`	set/override config option
`--version`	output version information and exit
`-h --help`	display help and exit

Configuration

Some simple configuration files

- `[.]hgrc`
  Main configuration file; used to define user or project wide options

- `.hgignore`
  Identify files/directories that should not be version controlled within the repository

Configuration - .hgrc

- Mercurial uses a layered configuration scheme for managing repositories and clones
- Later configuration overrides earlier
- Most common paths:
  - `<install-root>/etc/mercurial/hgrc.d/*.rc`
  - `<repo|clone>/.hg/hgrc`
  - `$HOME/.hgrc`

Configuration - .hgrc

- **The resource configuration files are used to:**
  - Define basic settings (user name, …)
  - Configure custom behavior
  - Activate available extensions

Configuration - .hgignore

- File used to ignore common spurious files (*~, .pyc, tar.gz, etc.)
- Both regular expressions and globbing are supported
  - globbing uses shell file matching syntax
    `*.log`
  - Regexp uses standard perl regular expressions syntax
    `^[hc]at`

Mercurial – Getting Help

- `hg help`
  - Provides a list of available commands

- `hg help <cmd>`
  - Provides information about command `<cmd>`

- `hg help -v` (`-v` for `--verbose`) will print more detailed information about the specified command

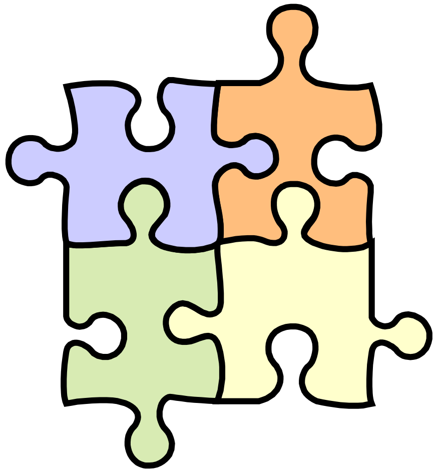
- *The built-in help will be used during the course to supplement the material presented*

Mercurial – Course Toolchain

- Mercurial
  - http://mercurial.selenic.com/wiki
- TortoiseHG (Windows)
  - http://mercurial.selenic.com/wiki
- SourceTree (Mac OSX)
  - http://www.sourcetreeapp.com/
- MercurialEclipse (Eclipse)
  - https://bitbucket.org/mercurialeclipse/main/wiki/Home

Use Case Models

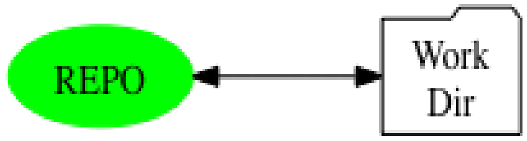
- Different use cases
  - Single User
  - Single Site, Multiple Users
  - Multiple Sites, Multiple Users
  - Service Center and Consulting support



Mercurial I

Single User

Case #1 – Single Developer Model



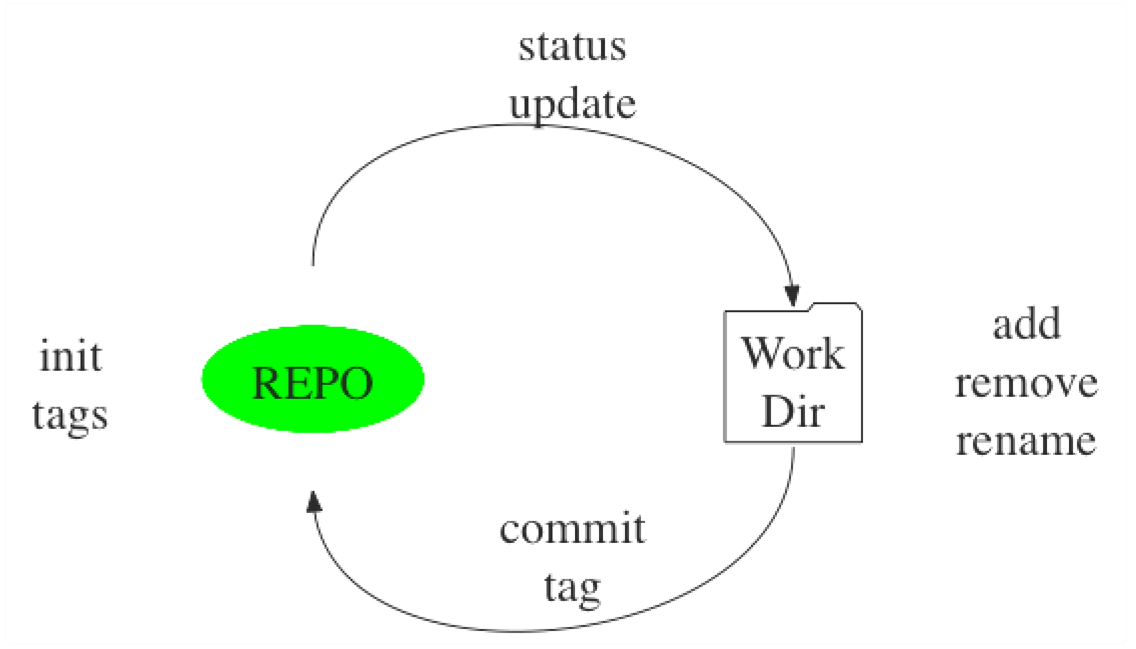
Single User

- Mercurial is used to manage source code, regardless of the number of developers

- Core feature set is not exclusive to distributed development

- Change tracking is essential, regardless of project type
  - Provides ability to track changes over time, with the possibility to revert to a particular revision

Single User

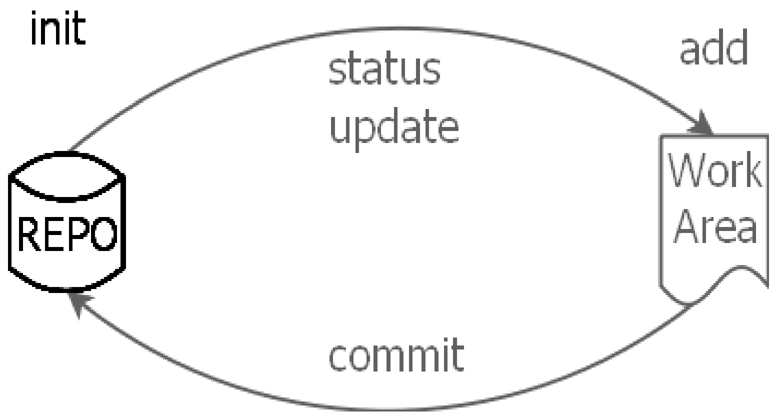


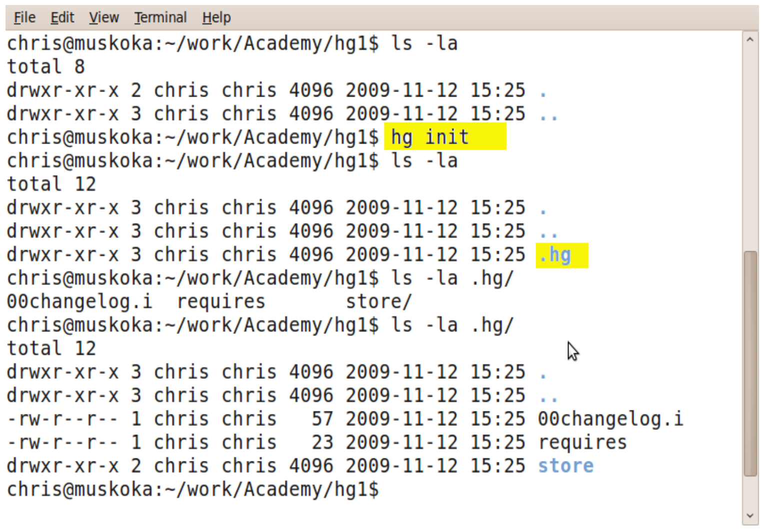
- Source code management for a single user is very similar across all the different VCS'
- A single user repository can be spawned into a multi-user repository at any time (just a matter of convention)

Commands - List

add	Add the specified files on the next commit
commit	Commit the specified files or all outstanding changes
diff	Show differences between revisions for files
init	Create a new repository in the given directory
log	Print the revision history of files or project
remove	Remove the specified files on the next commit
revert	Restore files to their checkout state
rename	Equivalent of copy + remove, will remember history
status	Show changed files in the working directory
serve	export the repository via HTTP
tag	Add one or more tags for the current or given revision
tags	List repository tags

Commands - init

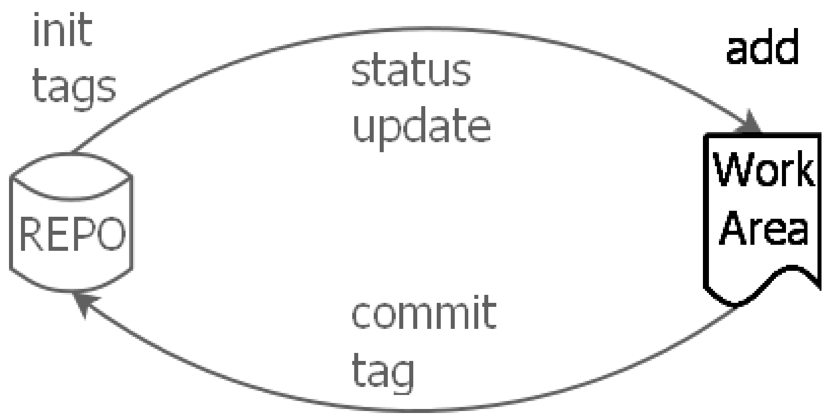




```
chris@muskoka:~/work/Academy/hg1$ ls -la
total 8
drwxr-xr-x 2 chris chris 4096 2009-11-12 15:25 .
drwxr-xr-x 3 chris chris 4096 2009-11-12 15:25 ..
chris@muskoka:~/work/Academy/hg1$ hg init
chris@muskoka:~/work/Academy/hg1$ ls -la
total 12
drwxr-xr-x 3 chris chris 4096 2009-11-12 15:25 .
drwxr-xr-x 3 chris chris 4096 2009-11-12 15:25 ..
drwxr-xr-x 3 chris chris 4096 2009-11-12 15:25 .hg
chris@muskoka:~/work/Academy/hg1$ ls -la .hg/
00changelog.i  requires       store/
chris@muskoka:~/work/Academy/hg1$ ls -la .hg/
total 12
drwxr-xr-x 3 chris chris 4096 2009-11-12 15:25 .
drwxr-xr-x 3 chris chris 4096 2009-11-12 15:25 ..
-rw-r--r-- 1 chris chris   57 2009-11-12 15:25 00changelog.i
-rw-r--r-- 1 chris chris   23 2009-11-12 15:25 requires
drwxr-xr-x 2 chris chris 4096 2009-11-12 15:25 store
chris@muskoka:~/work/Academy/hg1$
```

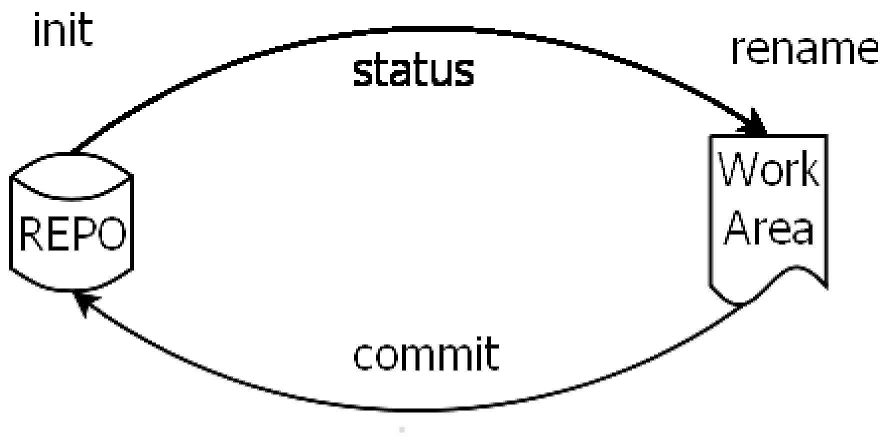
- Initialize a Mercurial repository; this creates the .hg directory for metadata
- By default, the current working directory becomes a repository
- Files and directories must be add(ed) to the repository to begin tracking

Commands - add



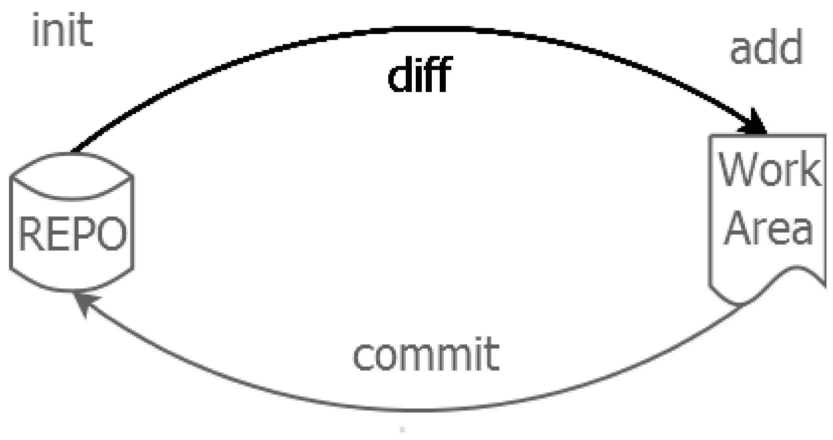
- Schedule existing working file(s) to be added to the repository upon next commit

Commands - status



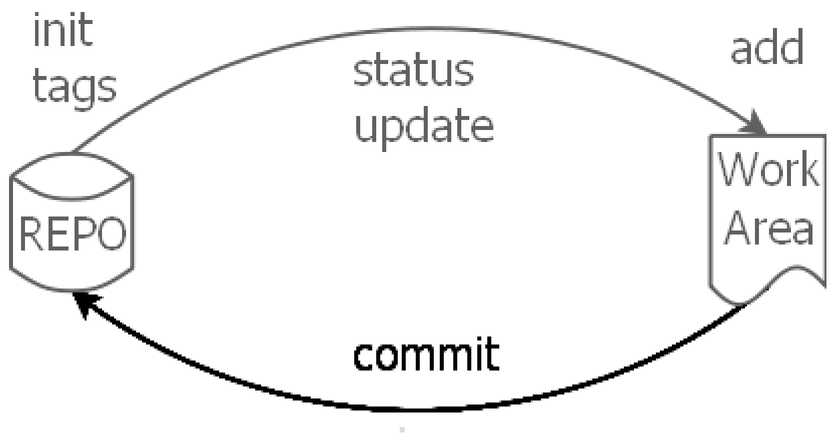
- Show the change status of the working files
- Some of the common codes showing the status of files:
  - M = modified
  - A = added
  - R = removed
  - ! = missing (deleted by non-hg command, but still tracked)
  - ? = not tracked

Commands - diff



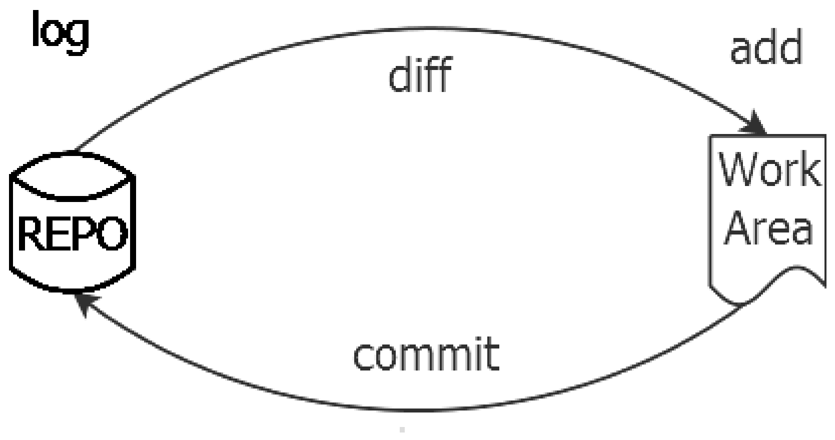
- Show differences between revisions for the specified files
- Use -c parameter to show differences in a changeset

Commands - commit



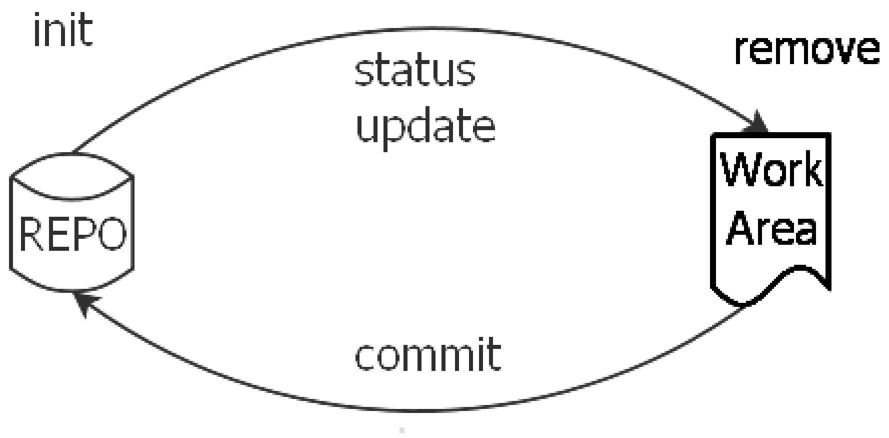
- Commit scheduled changes to the repository, creating a changeset
- Use the optional `-m` parameter to specify the commit message directly on the command line

Commands - log



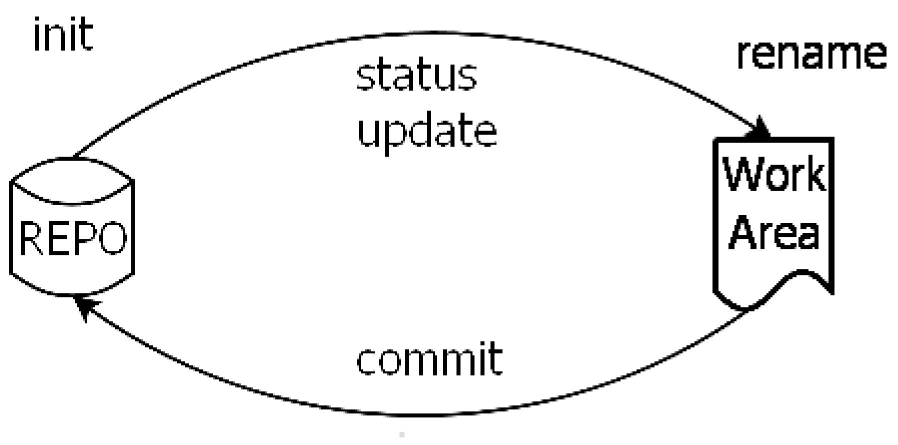
- Show revision history of entire repository or files
- Use -f to follow files across copies and renames

Commands - remove



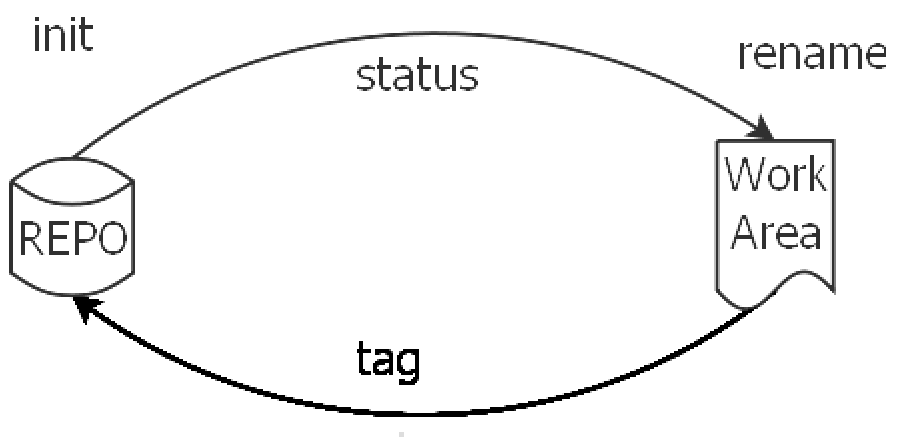
- Schedule file(s) for removal from the repository
- The file(s) remain in the project history
- File(s) will be deleted from work area, if still present

Commands - rename



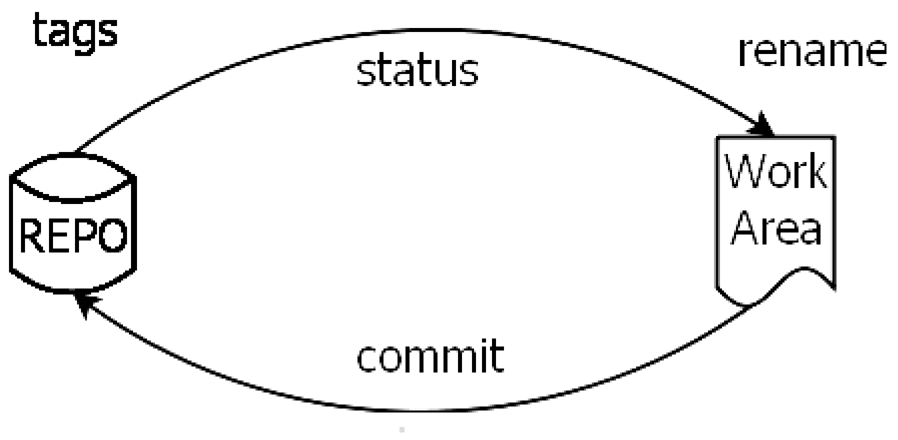
- Rename a file to a new name
- Associates the history of the original file to the renamed file
- Use --after for a rename that already occurred.

Commands - tag



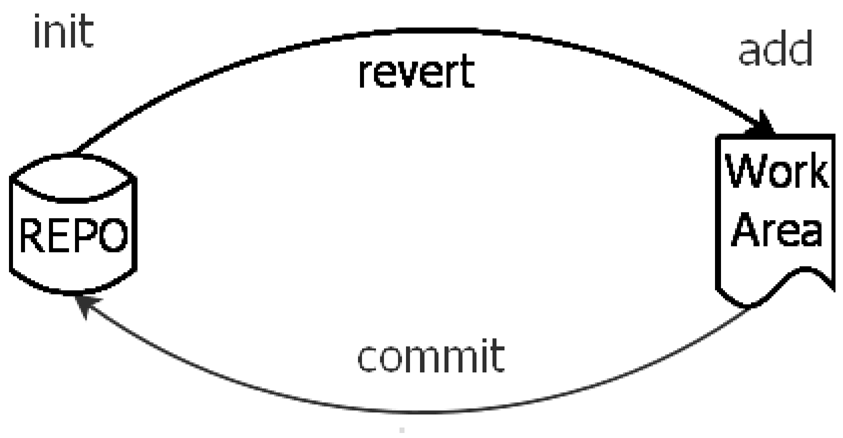
- Apply a named Tag the specified changeset revision
  - The parent of the working directory, by default
- Tag information is tracked in a repository text file: `.hgtags`
- Handy for multi-user projects
- See also: bookmark

Commands - tags



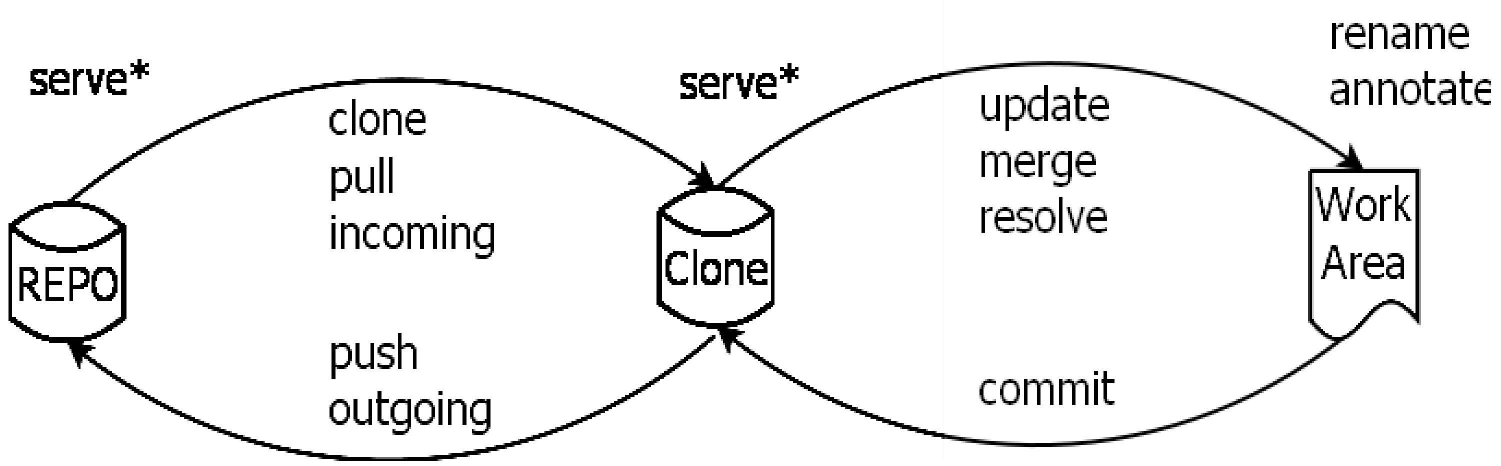
- List the repository tags existing in the project

Commands - revert



- Discard current changes and revert to checkout state
- Handy when the current changes are too broken to be useful
- Also used to revert to older revisions

Commands - serve



- Start an adhoc web service to export one repository for viewing in a web browser *(any repository can be served)
  - Handy alternative to `hg view` (especially when accessing across WAN)
- By default, uses the port 8000
- Connect to: `http://<server>:8000` in your web browser

Best Practices

Commit messages:

- For project work, the message should include requirements/work packages references,
  - to make them searchable later, and
  - to find more information about a changeset
- `-m “PSFG-2: Correct block time calculation bug when trip spans DST”`

Best Practices

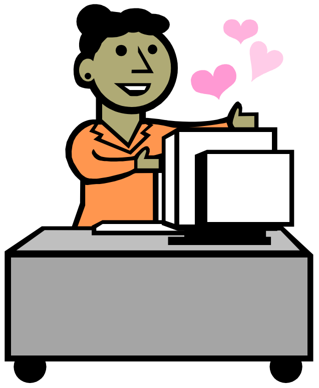
Changesets should be for atomic requirements (by convention)

- It must be possible to know which changes were made together to be able to revert
- Therefore, a changeset should address only *one* bug or *one* requirement
- If multiple tasks must be worked on concurrently, use multiple clones (seen later)

Exercise 1

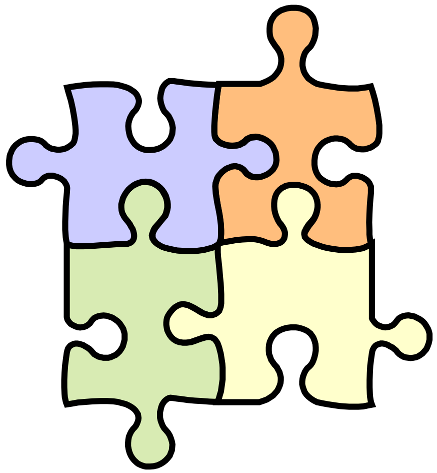
Exercise 1 summary

- Questions and answers



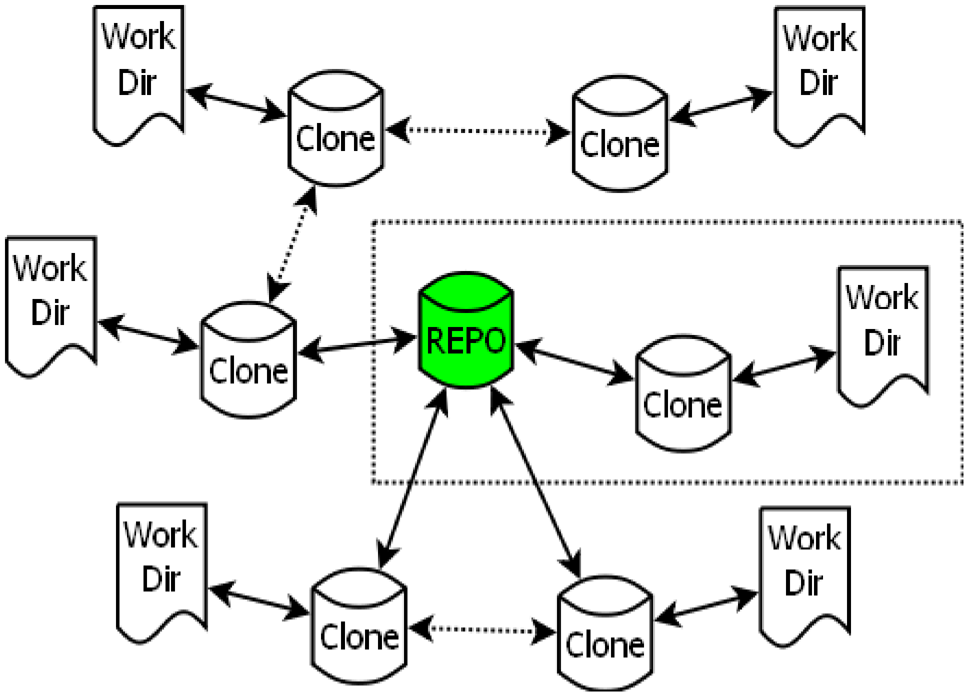
Use Case Models

- Different use cases
  - Single User
  - Single Site, Multiple Users
  - Multiple Sites, Multiple Users
  - Service Center and Consulting support

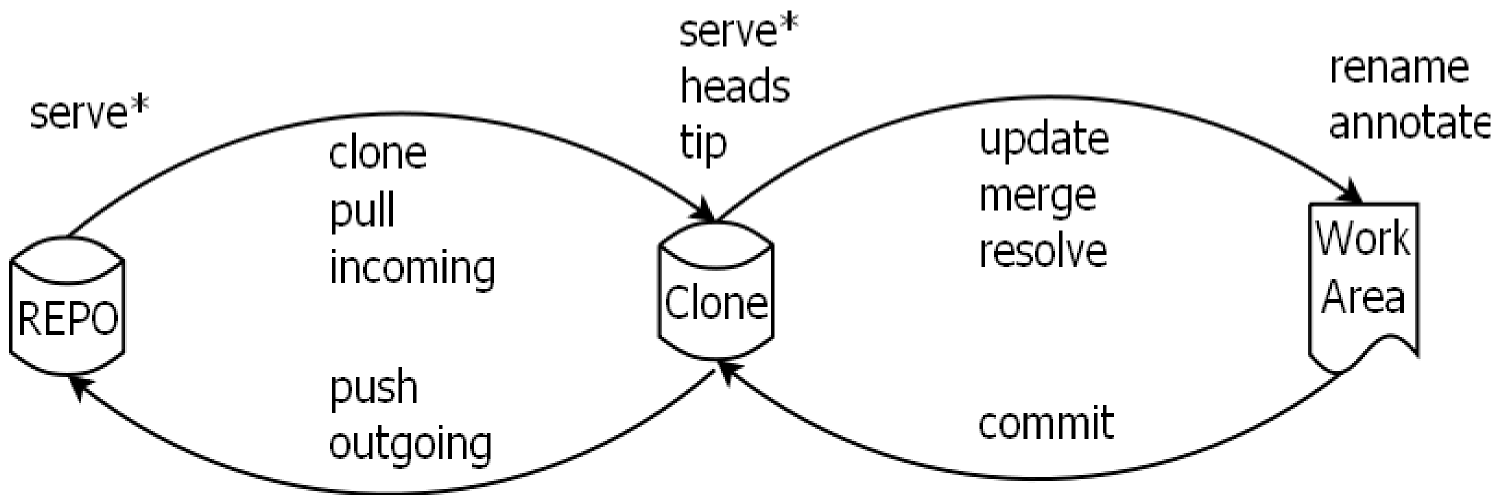


Multiple Developers

Case #2 – Multi-Developer Environment



Context Diagram



- In a multi-user configuration, each user has a clone of the repository, instead of working with the repository itself
- Sets of changes are shared with other users either directly or via the main repository

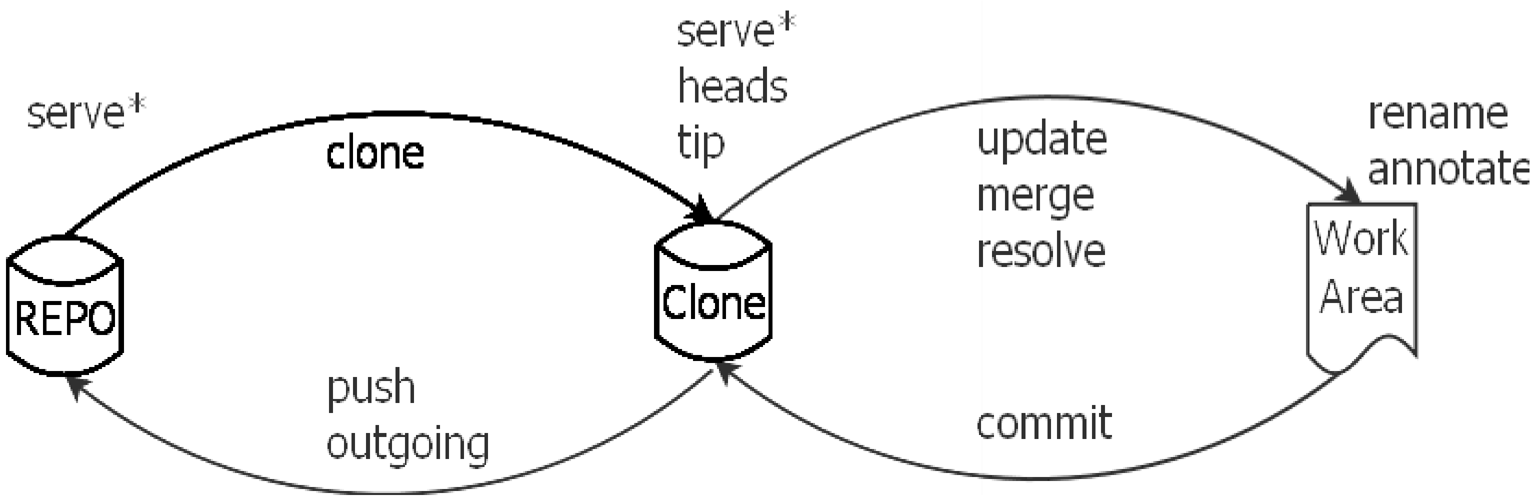
Multiple Users

- In a multi-user project, it is common practice to share at least one (1) central repository
- This ensures that all developers use project code taken from the same family
- Multiple repositories could be used as an alternative to code branches during a development project

Commands - List

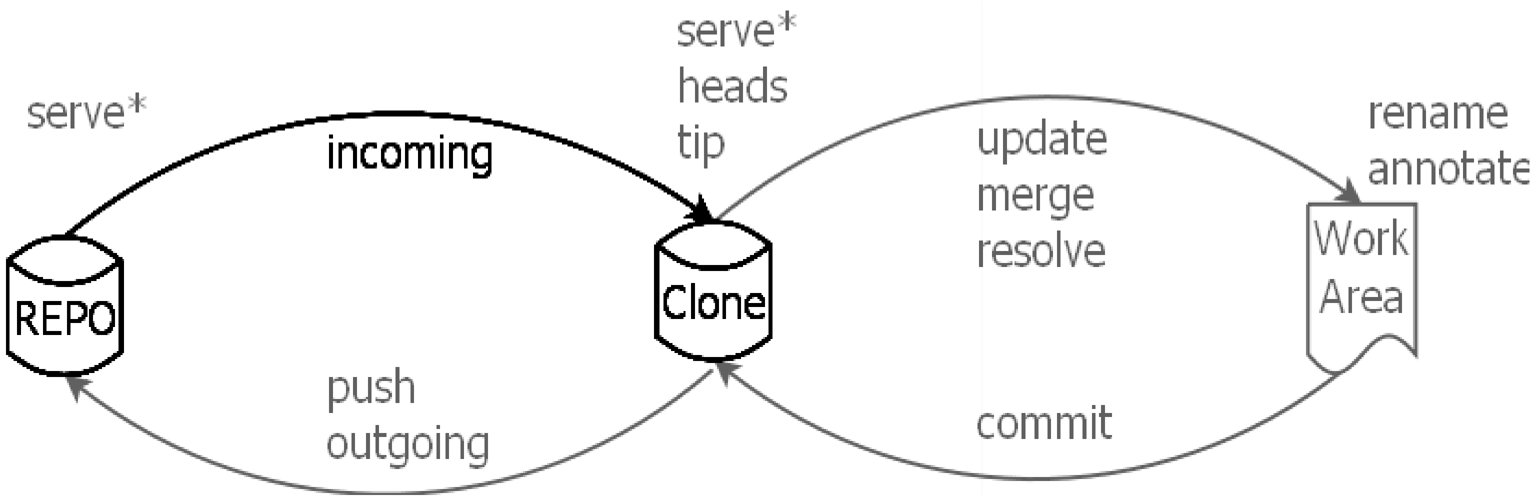
annotate	show changeset information by line for each file
clone	make a copy of an existing repository
heads	show current repository heads or show branch heads
incoming	show new changesets found in source
merge	merge working directory with another revision
outgoing	show changesets not found in destination
pull	pull changes from the specified source
push	push changes to the specified destination
resolve	retry file merges from a merge or update
update	Update working directory

Commands - clone



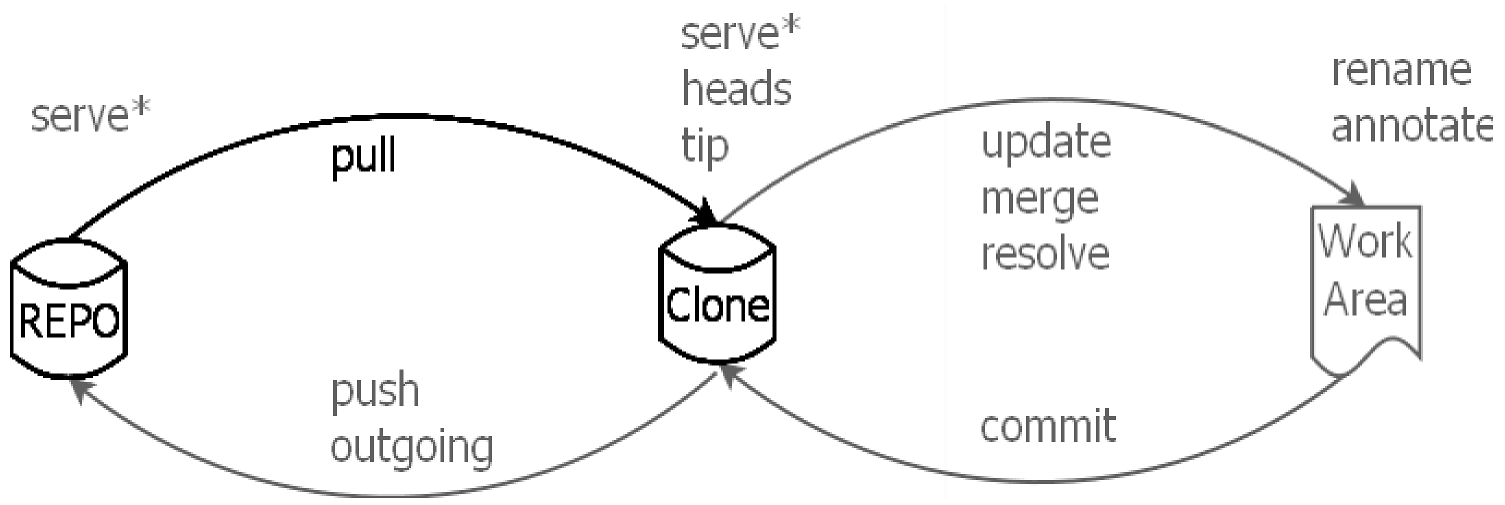
- Create a replica of a repository
  - Once the clone is created, a connection to the source repository is only required when synching with other project members
- Clones can be made to local or remote locations (using SSH)

Commands - incoming



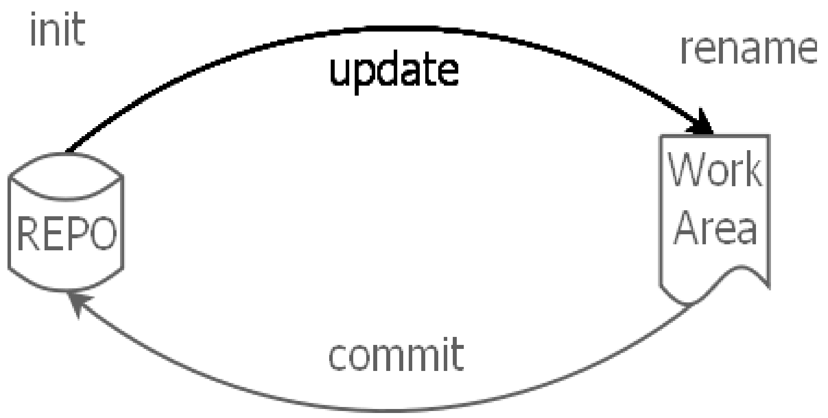
- Show a list of the changesets that would be pulled from the source repository
- Often used to see what needs to be merged, prior to performing a push

Commands - pull



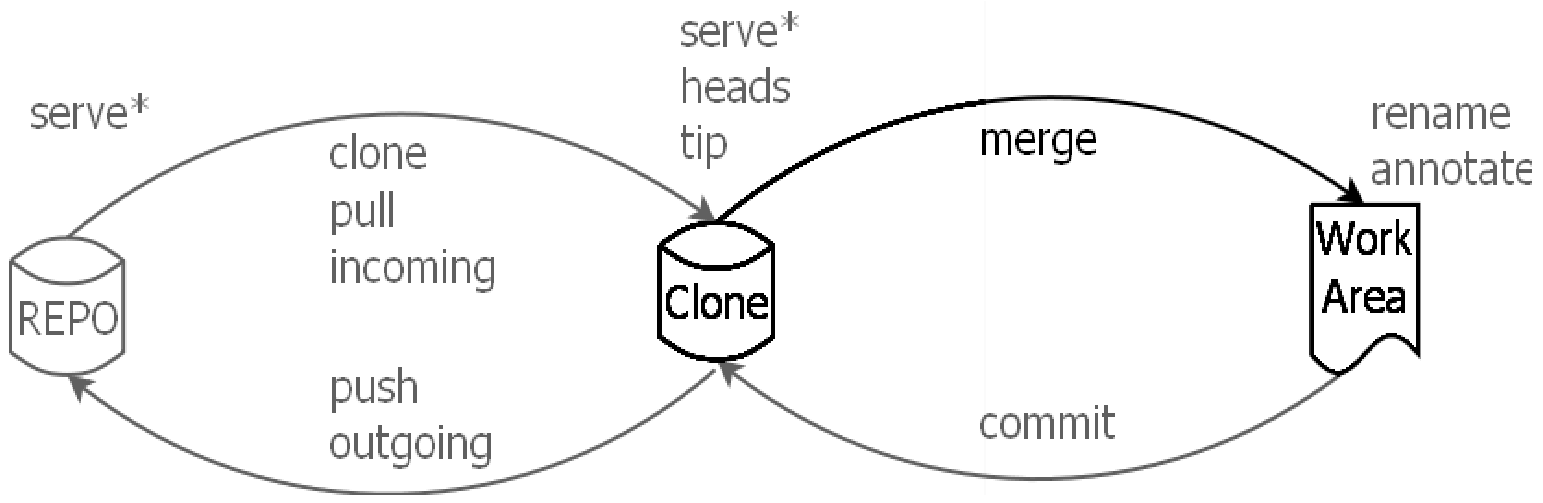
- Pull the remote changesets into the local repository
- Does not automatically update the local working directory (could use `-u` option to combine pull + update)
- Can only pull into an existing clone from the repository family

Commands - update



- Update the working directory to a specified revision
    - By default, updates to the `tip` revision
- `-C` option provides a clean move, overwriting modified files in the working directory
- Allows to move between revisions easily; especially useful when tracking down regression bugs

Commands - merge

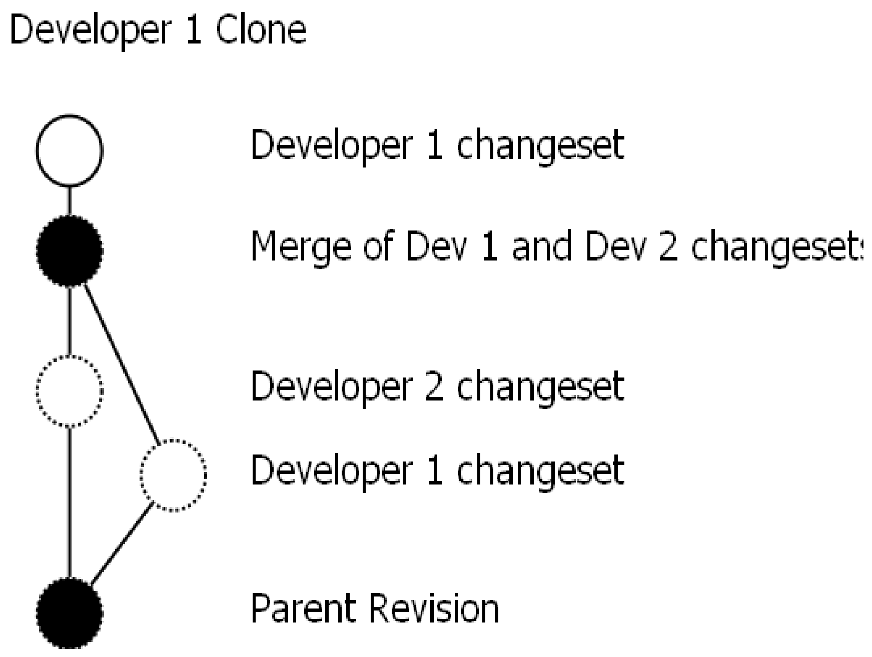


- Merge the current working directory with another version

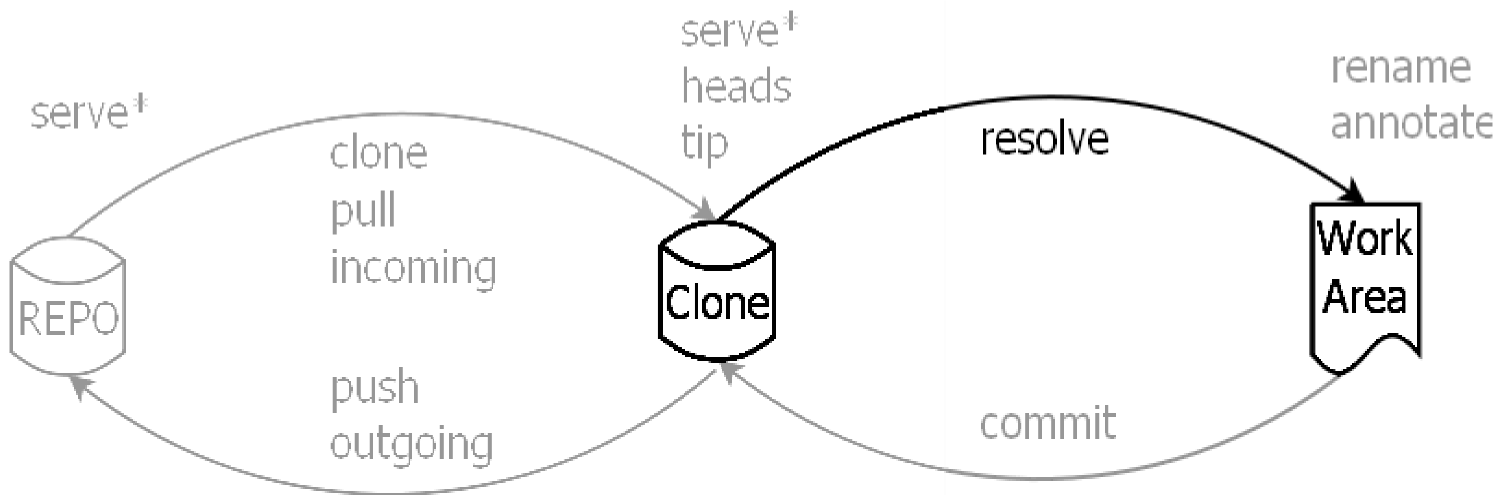
- Merging is a core concept in Mercurial, and is done frequently

Commands - List

- Mercurial automatically decides if a merge or an update is required after a pull
- Once the merge is completed successfully, a commit is required before making any further changes
- The merge becomes a changeset, capturing the results of the merge



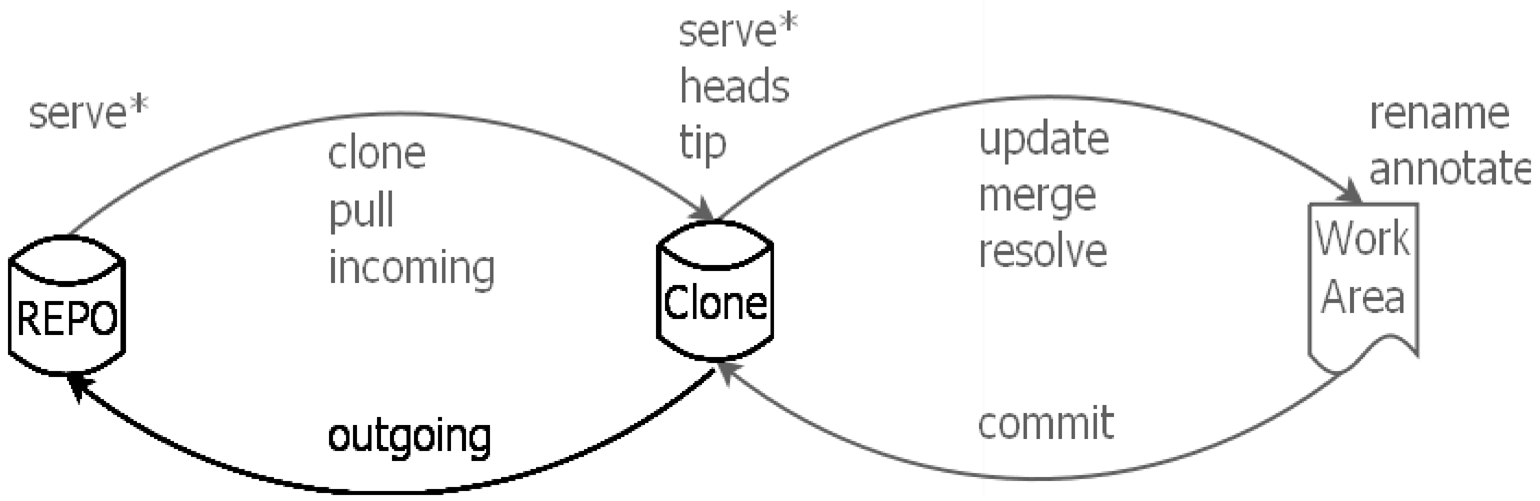
Commands - resolve



- Use to mark conflicting files as resolved during a merge
- Normally a merge tool is started automatically, e.g. Meld
- --list option shows which files require merging

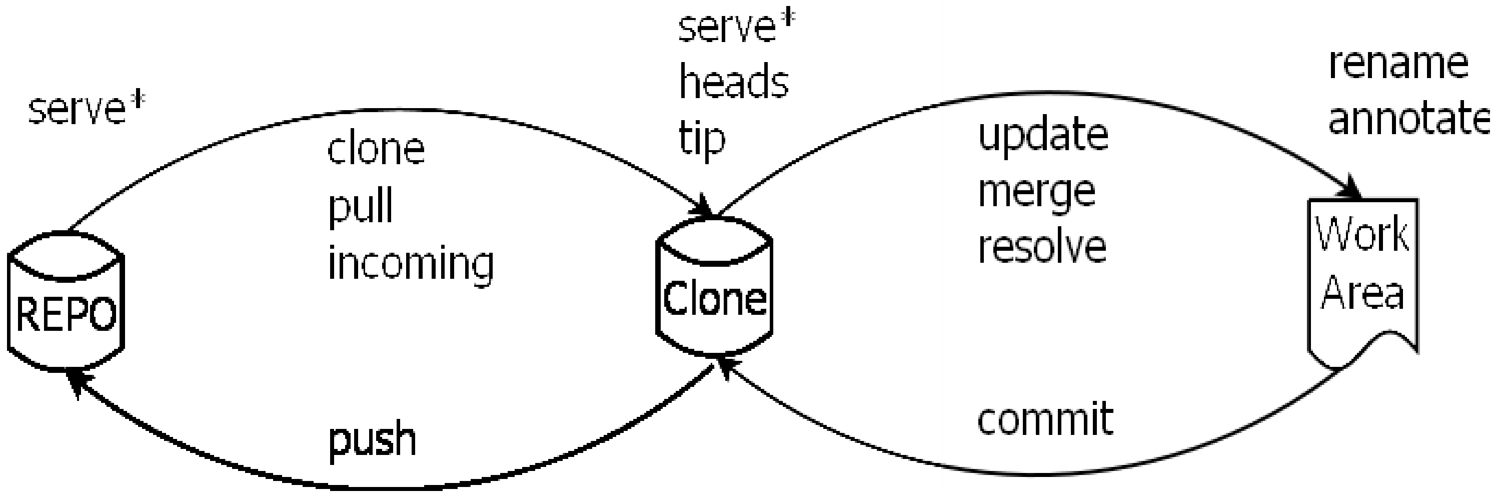
JEPPESEN

Commands - outgoing



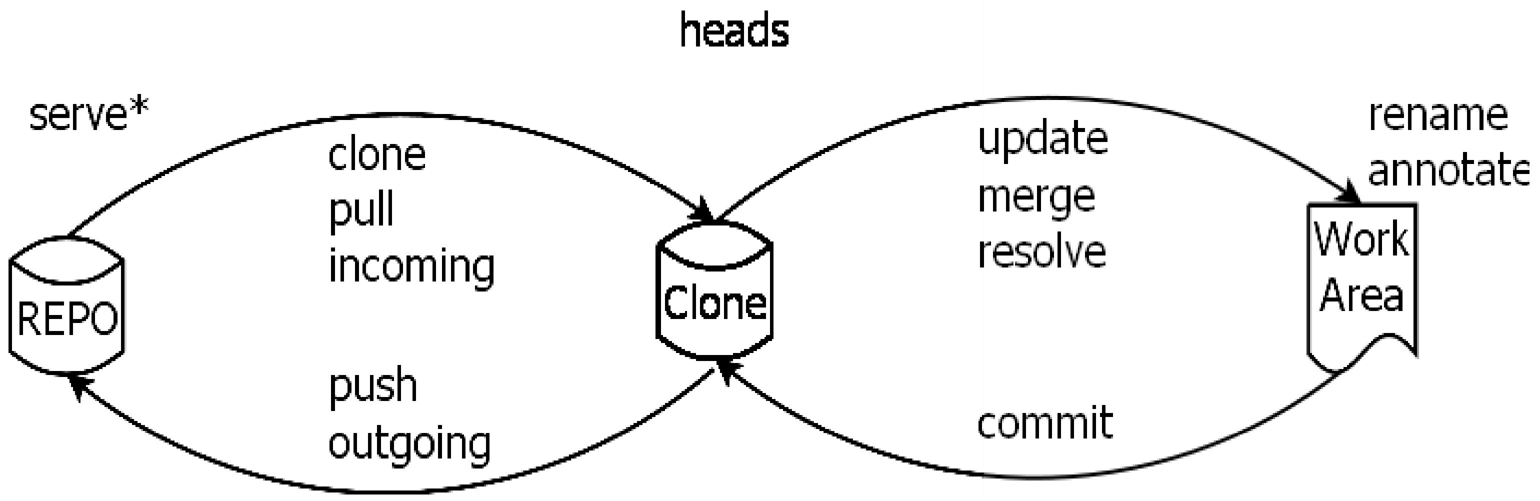
- List the local changesets to be synched at the next push to the specified repository
- Useful as a “test push”

Commands - push



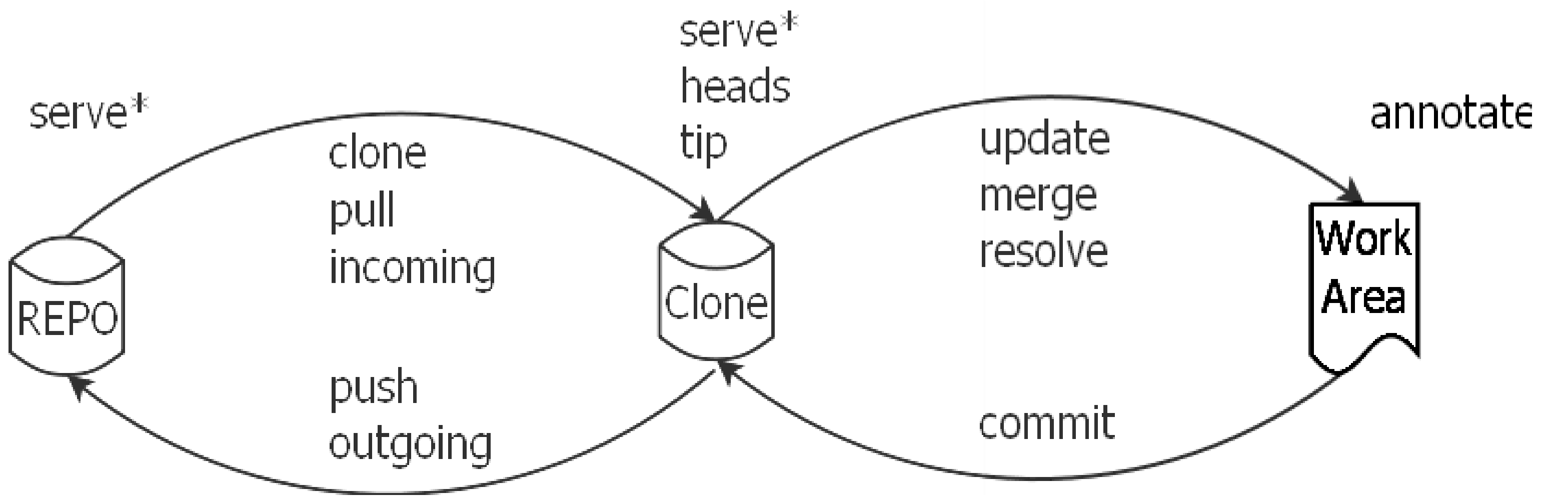
- Push the local changesets to the remote repository
- Changesets can be pushed to any repository derived from the common family tree
- By default, changes are pushed to the repository the clone was made from – shown in the file `<clone>/.hg/hgrc`
- NEVER use the `-f` option (force)

Commands - heads



- Show a list of the branch heads in the repository
  - Unless specifically planned, there should not be multiple heads in the repository

Commands - annotate



- Show changes per file on a line by line basis
- Helps in identifying when a particular change took place

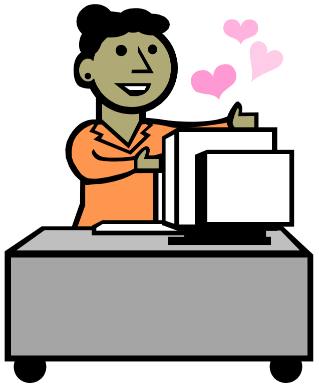
Best Practices

- All the rules from a single user environment still apply
- Changesets can be pushed in batches
  - It is not necessary to push after each commit
  - Conventional wisdom says:
    “Commit early, commit often”
- Avoid creating multiple heads, unless doing “branchy” development

Exercise 2

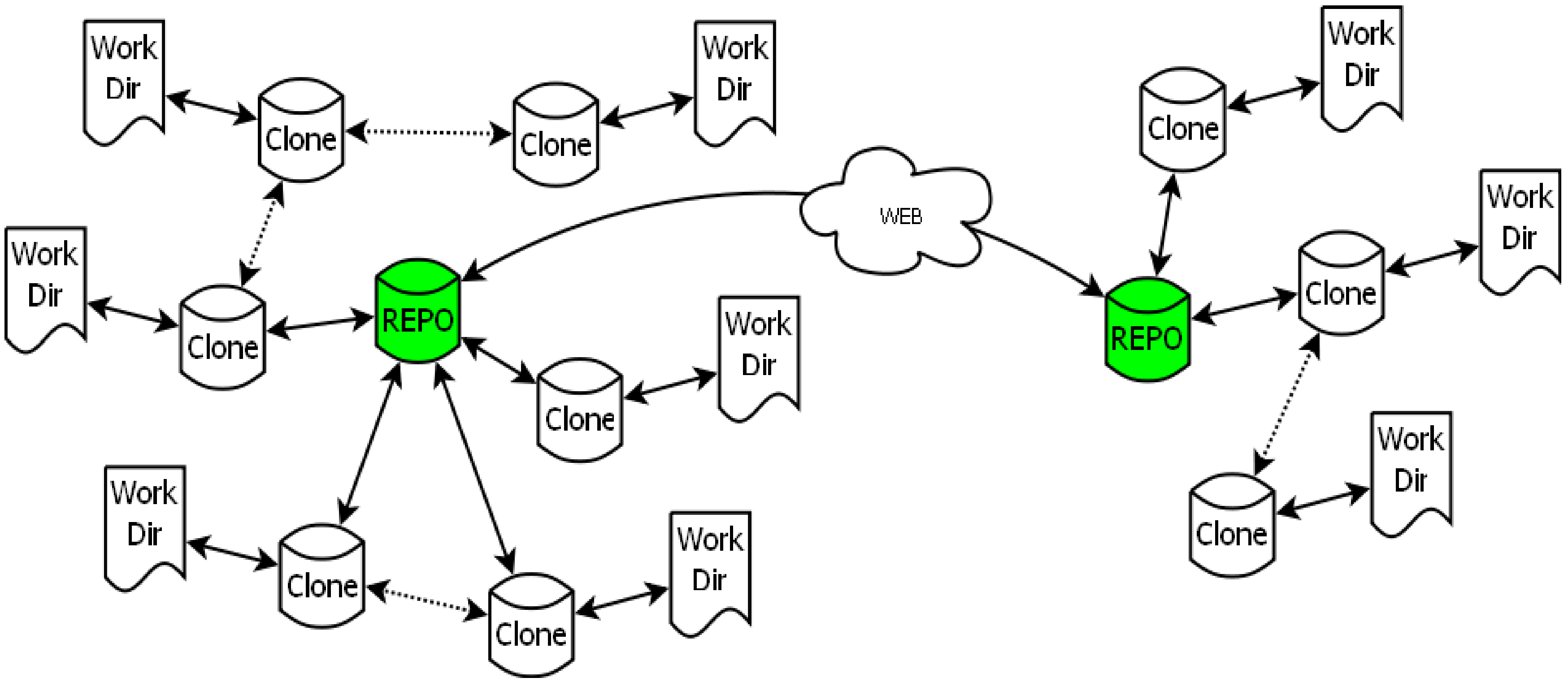
Exercise 2 summary

- Questions and answers



Multiple Sites

Case #3 – Multi-Site Environment



Multiple Users, Multiple Sites

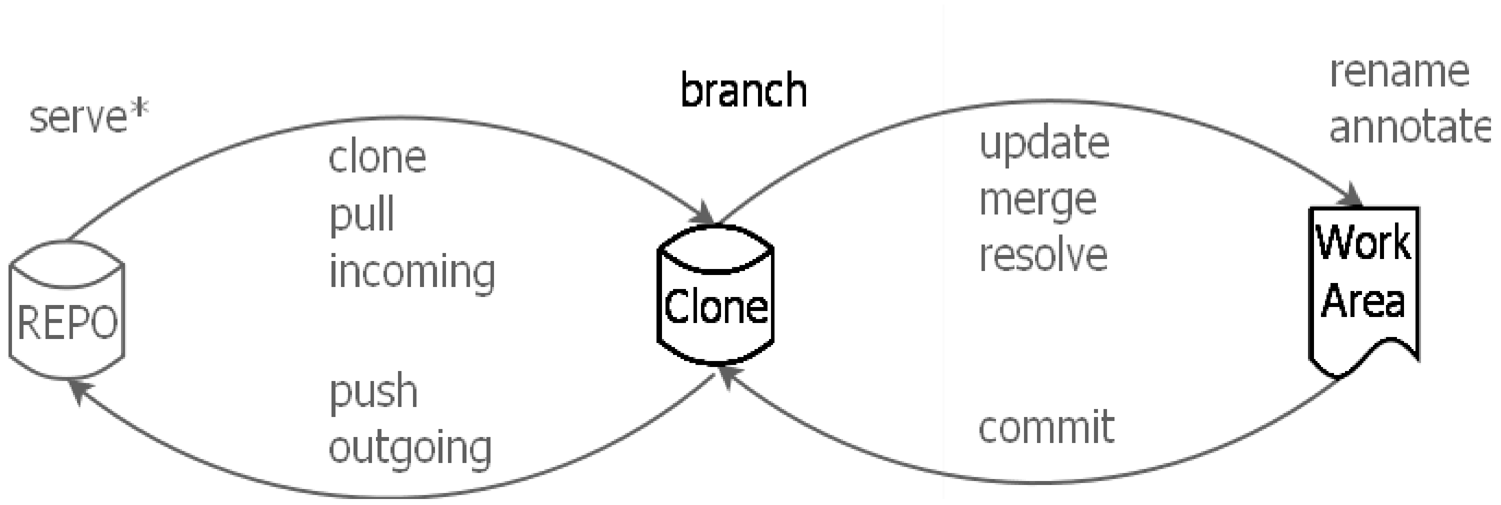
- In a multi-user, multi-site environment, developers sync their clones with a site-specific main repository
- An administrator usually handles synching between sites
- This setup is practical for performing interim deliveries and capturing “off-site” changes

Additional Commands

Some additional commands, though not specific to multi-site environments:

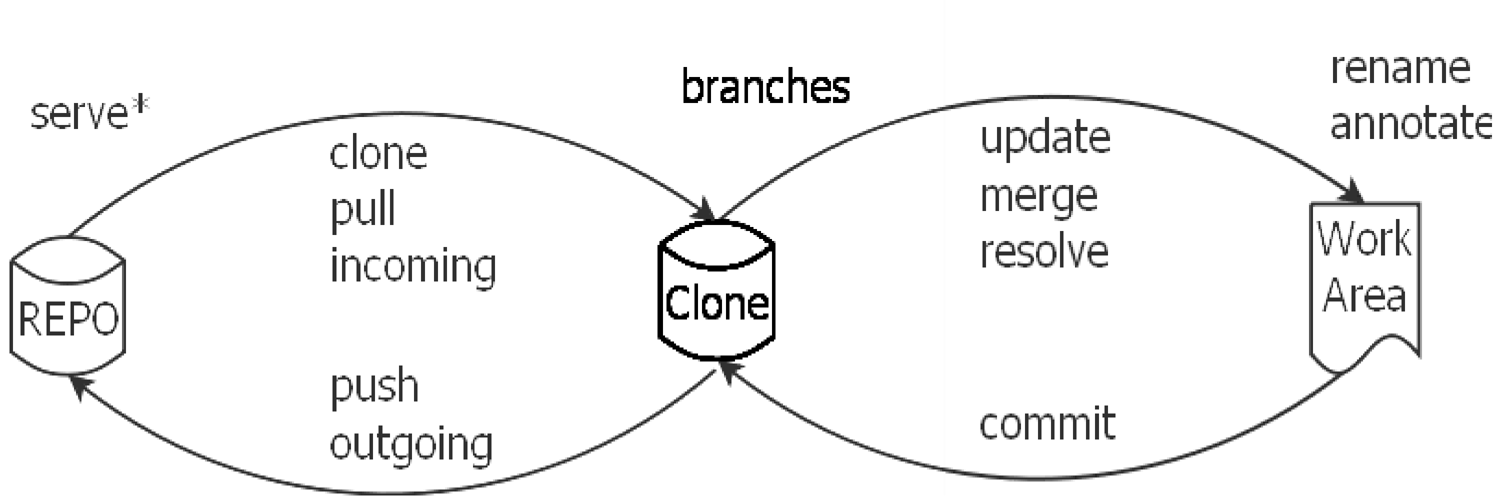
- branch
- branches
- export
- import
- graft

Commands - branch



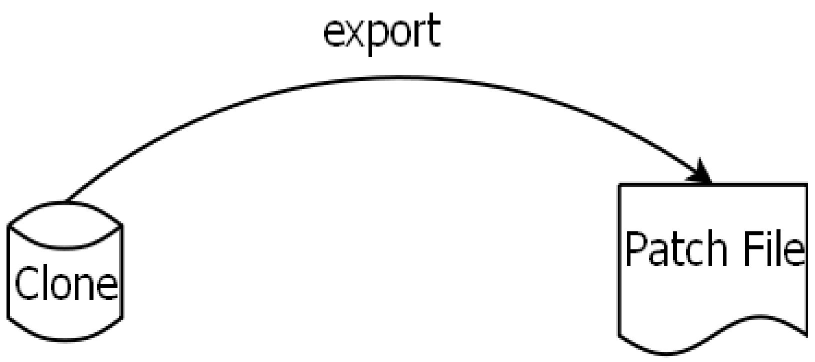
- `hg branch [NAME]`
- Display information about the current branch
- If a branch NAME is specified, instruct hg to create the branch (the next changeset commit will be placed into that branch)
- Also possible to use clones instead of branches (not recommended)

Commands - branches



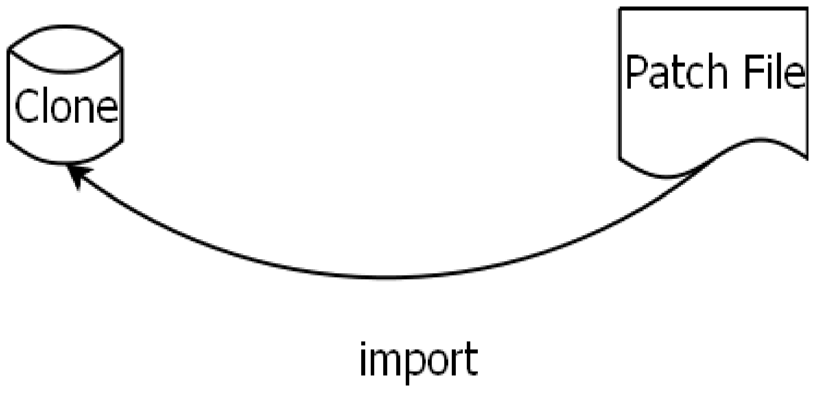
- Show the list of branches

Commands - export



- Export a set of changesets as a patch file
- Can be sent to someone working on the project for importing into their repository (see import)

Commands - import



- Import a patch file that was created using the hg export command
- Patch must be from same project tree

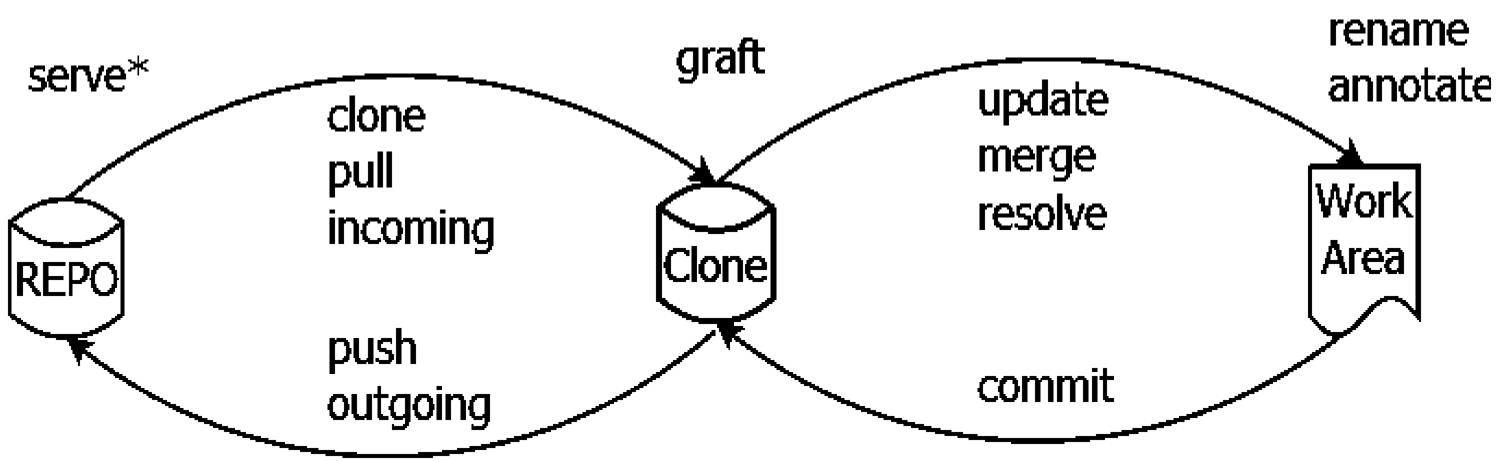
Export / Import - Best Practices

- For deployment during projects, allows better tracking of on-the-fly bug fixes at the client
- Changes made at client can always be captured and synched with the project code
- Internet connection to client not required; repositories can be kept in sync using clone carried on a portable device

Service Patch Mode

- Patches can be created from changesets in Mercurial using `export` and `import` commands
- These are simple text files containing all the changeset information required to import into another repository
- Service Manager/Consultant can build a changeset, create a patch file, and email it to the client for importing

Commands - graft



- Provides the ability to transplant changes from one named branch to another
- Common when only some of the changes should be included, while skipping others
- For example, include a bug fix from default branch in release branch

Extensions - List

- Mercurial has many extensions available
- Activate them using the .hgrc configuration file(s).
- Sample extensions:
  - extdiff
  - convert
  - rebase
  - color
  - graphlog (superfluous from 2.3, use log -G)
  - shelve (from 2.8)
  - hgk
  - purge
  - strip (from 2.8)

Extensions - hgk

- Enables graphical project view
- Requires some special configuration, since the view is not installed as part of Mercurial
  - This is normally done as part of Jeppesen environment configuration

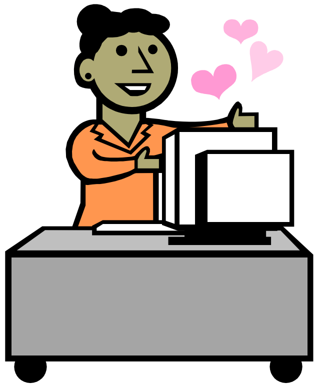
Extensions - extdiff

- Enable graphical diff tools to be used
  - Many different tools already available and configured at Jeppesen by default
- Meld, tkdiff, kdiff3 are all viable options

Exercise 3

Exercise 3 summary

- Questions and answers



Course Evaluation

Please take a few minutes to complete the evaluation form, it will help us improve the courses for you and your colleagues:

```
    Special> Academy> Course Evaluation
```

Are the exercise definitions too vague (too real-life), would you like them to be more exact and straight forward?

Would you like to have even more info on slides (for self studying) or would you be stressed about the time constraint?

The end

This was Mercurial I

Welcome back to Jeppesen Crew Academy!