Week 09M: Change Management

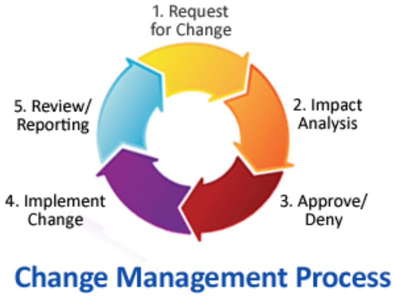
Thuy Vu

Reminders

- Assignment 8: Dynamic linking
  - `web.cs.ucla.edu/classes/winter17/cs35L/assign/assign8.html`
  - Time due 23:55 <u>this</u> Friday, March 10
- Assignment 9: Change management
  - `web.cs.ucla.edu/classes/winter17/cs35L/assign/assign9.html`
  - Time due 23:55 <u>this</u> Friday, March 17
- Assignment 10: Presentation
  - Thursdays of this and next week

Software Change Management

"the only constant thing in software engineering is change."



;

Change Description in Six Ws

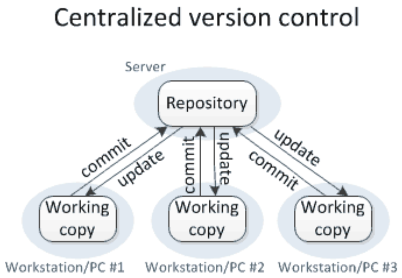
When	timestamps	How	to communicate changes (patch)
Who	made the change	What	is the change (= origin + diff)
Where	was the change made	Why	does it exist

`diff` and `patch` limitations

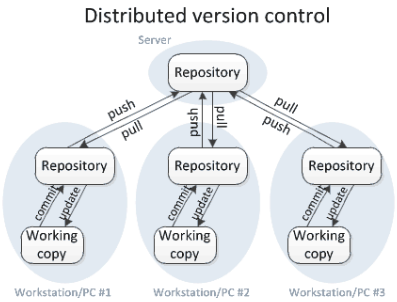
- `diff` needs the *exact* original copy
  - no matter how big/small the change is
- too simple (old v.s. new)
- not "scalable"

version control tools: Apache Subversion, git, Bazaar

Centralized vs Distributed Version Control



- one central repository
- changes in once place
- e.g.: Subversion
- simple, changes immediately seen
- no backup!

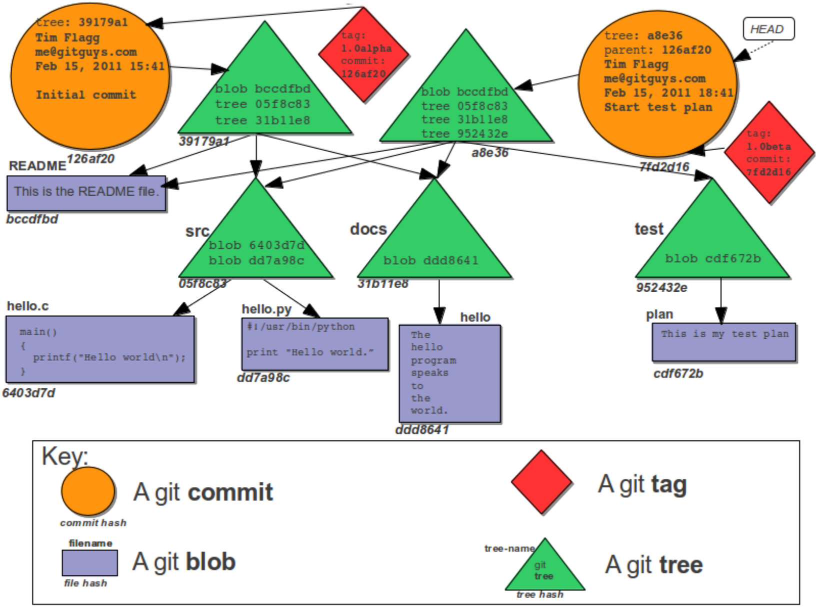


- multiple repositories
- you push & commit
- others pull & update
- e.g.: git, Bazaar, Bitkeeper
- can do offline; fast;
- space; download time;

Terminology

1. repository – files and folders of a full history and versions (database of changes)
2. working copy – a local copy of a local repository
3. check-out – making a local working copy from repository
4. commit – writing changes made in working copy to repository
5. git objects – 4 types
   - blob (binary-large-object) – content of a file (w/o name, timestamp...)
   - tree – a folder with filenames, each is (name+blob) / a tree / soft-link
   - commit – points to the top-level tree of the `"git commit"`-ed project
   - tag – name to a commit object

git's Objects



git's Workflow

A file can be in one of three main states

1. committed – is safely stored in the local repository

git's Workflow

A file can be in one of three main states

1. committed – is safely stored in the local repository

3. modified – has been changed by you but yet committed to the repository

`git`'s Workflow

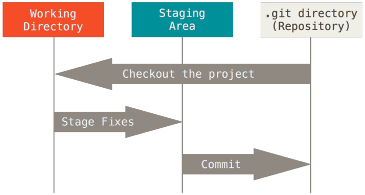
A file can be in one of three main states

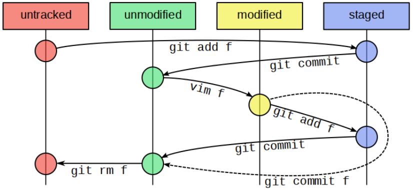
1. committed – is safely stored in the local repository
2. staged – is marked to be committed in the next commit snapshot
3. modified – has been changed by you but yet committed to the repository

git's Workflow

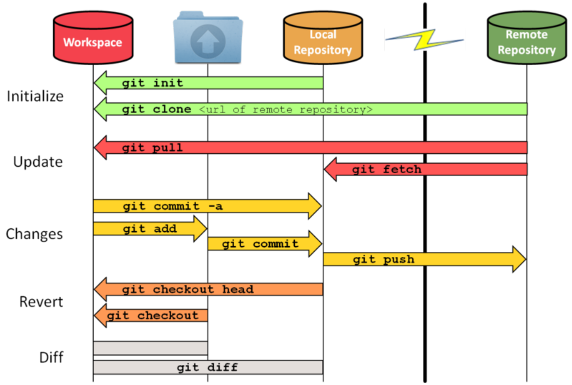
A file can be in one of three main states

1. committed – is safely stored in the local repository
2. staged – is marked to be committed in the next commit snapshot
3. modified – has been changed by you but yet committed to the repository





git's Workflow



git's Commands

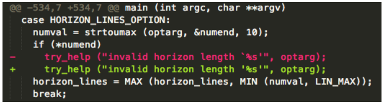
1. `git init` creates new respository
2. `git clone` get a copy of an repository
3. `git add` add files to the index
4. `git commit` add changes to repository
5. `git help` `git status` `git log` `git show` `git diff`
6. `git checkout` checkout a specific version
7. `git reverse` reverse a commit using patch

Lab 4 – Managing a Backported Change

-
  ```
  me$ diff .  -
  diff:  cannot compare `-' to a directory
  ```
  ⇒ maint: quote 'like this' or "like this", not `like this'
- Backporting is to apply a patch to an older version of the software than it was initially created for.

the steps

1. get a copy of the repository
2. get a log of changes to master branch
3. generate a list of tags
4. find the commit and generate a patch for it
5. check out version 3.0 from local repository
6. apply the patch
7. check the status
   - untracked files
8. reverse all the changes to files other than `*.c` files
9. undo changes to `*.c` files not related to character string constants
10. rebuild as instructed in `README-hacking`
11. `just_built/diff -pru diffutils-3.0 diffutils-3.0-patch`



```
@@ -534,7 +534,7 @@ main (int argc, char **argv)
   case HORIZON_LINES_OPTION:
     numval = strtoumax (optarg, &numend, 10);
     if (*numend)
-      try_help ("invalid horizon length `%s'", optarg);
+      try_help ("invalid horizon length '%s'", optarg);
     horizon_lines = MAX (horizon_lines, MIN (numval, LIN_MAX));
     break;
```

Homework 4 – Verifying and Publishing a Backported Change

1. create a new branch named "quote" of version 3.0
   - `git checkout v3.0 -b quote`
2. apply patch from the lab on this branch
   - `patch -pnum < quote-3.0-patch.txt`
3. `emacs`
4. commit changes to the new branch
5. generate a patch of your changes for your partner
   - `git format-patch`
6. test your partner's patch
   - check out version 3.0 into a partner branch
   - apply patch with `git am < formatted-patch.txt`
   - then, `make check`